SIEMENS

WinCC

组态手册

第一册

订货号：6AV6 392-1CA05-0AH0
C79000-G8276-C164-01

发行：1999 年 9 月

WinCC、SIMATIC、SINEC、STEP 是西门子注册商标。

本手册中所有其它的产品和系统名称是（注册的）其各自拥有者的商标，必须被相应地对待。

（我们已检查了本手册的内容，使其与硬件和软件所描述的相一致。由于不可能完全消除差错，我们也不能保证完全的一致性。然而，本手册中的数据是经常规检查的，在以后的版本中包括了必要的修正。欢迎给我们提出建议以便改进。）

改变的技术数据

C79000-G8276-C164

目录

前言

本手册的目的

本手册通过下列章节来介绍 WinCC 的组态选项 ：

本手册采用印刷版和电子手册形式出版。

目录表和索引可以帮助您快速找到需要的信息。而且，在线文件包含了附加的搜索功能。

使用本手册的先决条件

具有 WinCC 基础知识（例如，通过学习使用入门) 或具备使用 WinCC 的实际组态经验。

附加支持

如果存在技术问题，请与 当地 Siemens 分公司联系。

此外，您可以拨打热线电话
+49 (911) 895-7000 (传真 7001)

关于 SIMATIC 产品的信息

可以在 CA01 目录中获得关于 SIMATIC 产品的最新信息。可通过下列 Internet 地址访问此目录：

http://www.ad.siemens.de/ca01online/

此外，SIMATIC 客户支持提供了最新的信息并提供下载。从下列 Internet 地址可查找有关技术咨询的解答：

http://www.aut.siemens.de/support/html_00/index.shtml

1 组态手册

组态手册是 WinCC 文档的一部分，它主要是有关 WinCC 在项目中的实际应用。

引言

近几年来，对监视和控制生产过程以及对生产数据进行归档和进一步处理的系统要求已在急剧增加。为了满足这些新的要求，新的 HMI 系统在过去几年的基础上已有所提高。
WinCC 就是其中的一个新系统。无论是从功能性、开放性和现代化程度而言，WinCC 毫无疑问都是最优的。
老一代的 HMI 系统通常仅提供一种完成特定任务的方案。而用 WinCC，用户几乎总能有大量不同的选择方案来完成任务。本组态手册的编制是为了确保用户能根据项目的性能和所需的组态工作量采用最佳的解决方案。
本说明旨在提供建议方案，以在工程项目中能更有效地使用 WinCC。
这些建议方案已在 WinCC 实例项目中采用。这些实例项目随 WinCC 光盘一起提供。用户可以直接在自己的项目中使用这些建议方案，以节约组态过程中的宝贵时间。

1.1 组态手册 - 有关结构和应用的注意事项

要求

在使用本组态手册工作前，应已具备一些使用 WinCC 的实际经验。WinCC 的初学者将发现入门手册是一本学习并熟悉 WinCC 的理想的起点型手册。入门手册通过较小的演示实例说明了主要的讨论对象和功能。本组态手册是对 WinCC 帮助系统(在线和文档)的补充。如果对象的特性、属性和其它论题在组态手册中未进行说明，则在帮助系统中可找到其描述。

内容和结构

本手册划分为以下几部分：

- WinCC - 概念本节包含 WinCC 系统的常规信息。
- 组态 - 常规主题本节包含有关规划和有效管理 HMI 项目的常规信息和特定信息。
- 启动实例本节包含有关开始运行本手册中所创建实例的信息。
- WinCC C 课程本节包含 WinCC C 课程。对于入门者，描述了使用 WinCC 脚本语言的主要规则。C 语言高手可以参阅有关 WinCC 开发环境特性的描述。
- 变量组态本节描述了 Project_TagHandling 实例。在本实例项目中，描述了变量的常规处理和简单的输入/输出元素。
- 画面组态本节描述了 Project_CreatePicture 实例。在本实例项目中，描述了 WinCC 画面的常规处理。
- WinCC 编辑器本节描述了 Project_WinCCEditors 实例。在本实例项目中，描述了变量记录、报警记录和报表编辑器。
- 用户归档本节描述了 Project_UserArchive 实例。在本实例项目中，描述了用户归档编辑器。
- 新功能描述本节描述了组态已添加到 WinCC V5 中的分布式系统的选项。
- 多客户本节通过实例项目描述了多客户项目类型的应用。
- 分布式服务器本节通过实例项目描述了分布在多个服务器上的 WinCC 项目的创建。
- 冗余本节通过实例项目描述了冗余服务器对的组态。
- 附录本节有关其它的各种主题。其中这些主题来源于 WinCC 解决方案和 WinCC 提示技巧。

规定

本组态手册使用了下列规定：

规定	描述
	表示使用鼠标左键进行操作。
R	表示使用鼠标右键进行操作。
D	表示双击鼠标左键进行操作。
斜体字	表示在 WinCC 环境中的术语，以及有关程序界面元素的术语。
斜体字，绿色	表示用户要遵守的操作顺序或条目(颜色只有在在线文档中才可见)。
蓝色	交叉索引(链接)用蓝色表示(颜色只有在在线文档中才可见)。

查找信息

在印刷成册的组态手册中，可以用下列方法查找信息：

- **目录表**列出了按主题编排的信息。
- **索引**列出了按关键字编排的信息。

在在线文档中，可以用下列方法查找信息：

- 目录标签列出了按主题编排的信息。
- 索引标签列出了按关键字编排的信息。
- 查找标签允许在整个文档中搜索字。

本手册中描述的实例项目可以直接从在线文档复制到您的硬盘驱动器中。

2 WinCC - 常规信息

2.1 WinCC - 概念

通常，从组态的角度上来看，在 WinCC 中有三种解决方案：

- 使用标准 WinCC 资源的组态
- 利用 WinCC 通过 DDE、OLE、ODBC 和 ActiveX 使用现有的 Windows 应用程序
- 开发嵌入 WinCC 中的用户自己的应用程序(用 VisualC++或 Visual Basic 语言)。

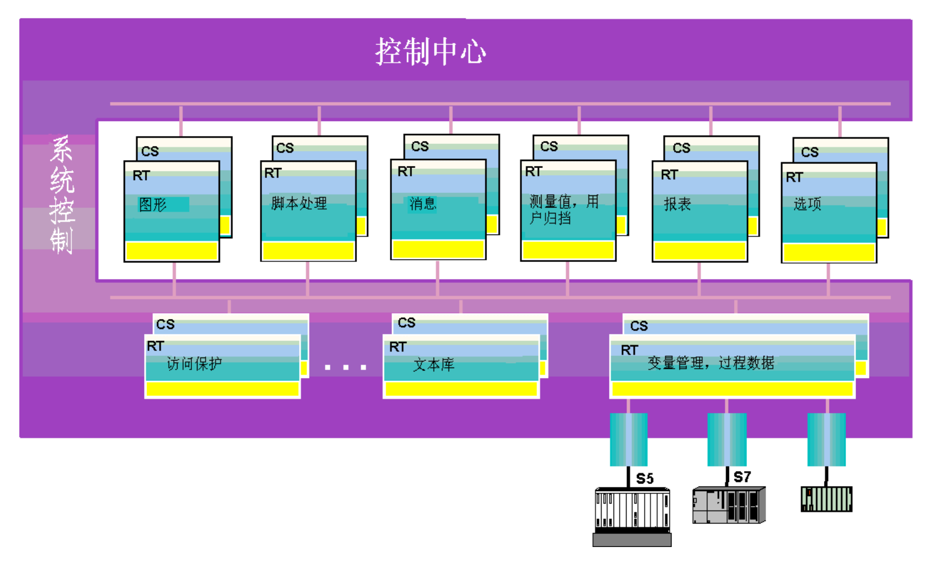
从某些方面看，WinCC 是进行廉价和快速的组态的 HMI 系统，从其它方面看，它是可以无限延伸的系统平台。WinCC 的模块性和灵活性为规划和执行自动化任务提供了全新的可能性。

操作系统：WinCC 的基础

WinCC 是基于 Microsoft 的 32 位操作系统(Windows NT 4.0)。该操作系统是 PC 平台上的标准操作系统。

WinCC 的模块结构

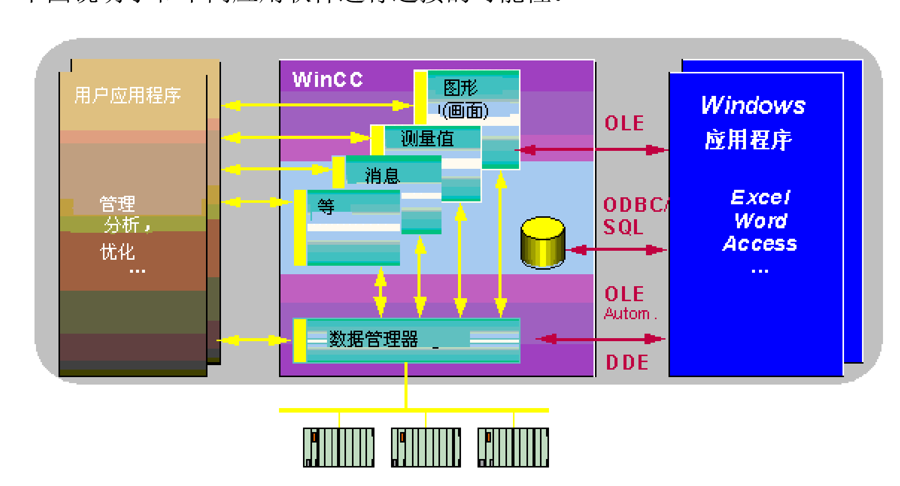
WinCC 为过程数据的可视化、报表、采集和归档以及为用户自由定义的应用程序的协调集成提供了系统模块。
此外，用户还可以合并自己的模块。



2.1.1 WinCC 接口

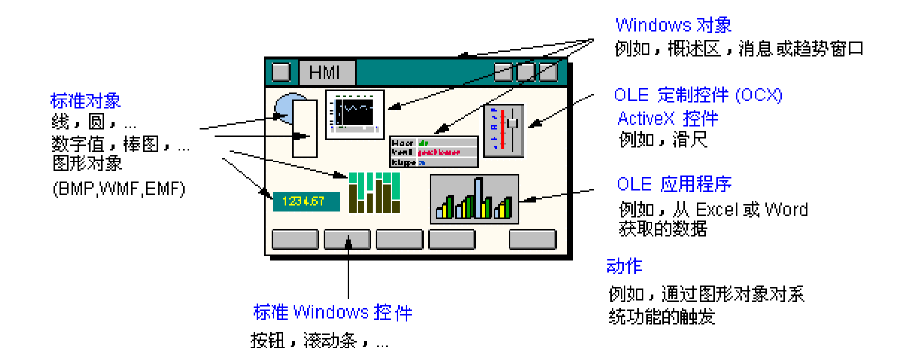
WinCC 的开放性

WinCC 对用户所添加的任何形式的扩充是绝对开放的。该绝对开放性通过 WinCC 的模块结构及其强大的编程接口来获得。
下图说明了和不同应用软件进行连接的可能性。



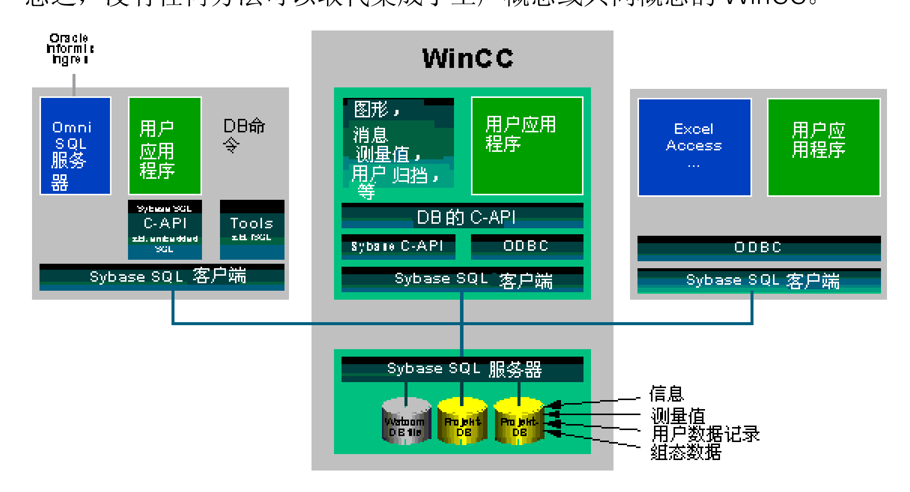
将应用软件集成到 WinCC 中

最重要的事实是 WinCC 提供了一些方法来将其它应用程序和应用程序块统一地集成到用于过程控制的用户界面中。
正如下图所示，OLE 应用程序窗口和 OLE 自定义控件(32 位 OCX 对象)或 ActiveX 控件可以集成到 WinCC 应用软件中，就好象它们是真正的 WinCC 对象一样。



WinCC 中的数据管理

在下图中，WinCC 组成了整个中心部分。该图显示了缺省数据库 Sybase SQL Anywhere 从属于 WinCC。该数据库用于存储(事务处理保护)所有面向列表的组态数据(例如变量列表和消息文本)，以及当前过程数据(例如消息、测量值和用户数据记录)。该数据库具有服务器的功能。WinCC 可以通过 ODBC 或作为客户通过开放型编程接口(C-API)来访问数据库。
当然，也可以将同样的权限授予其它程序。因此，不管应用程序是在同一台计算机上运行还是在联网的工作站上运行，Windows 电子表格或 Windows 数据库都可以直接访问 WinCC 数据库的数据资源。在数据库查询语言 SQL 和相关连接工具(例如 ODBC 驱动程序)的帮助下，其它客户端程序(例如 UNIX 数据库，如 Oracle、Informix、Ingres 等)也可以访问 WinCC 数据资源。当然，反之亦然。总之，没有任何方法可以取代集成了工厂概念或共同概念的 WinCC。



2.2 WinCC - 术语及其解释

本节包含了与 WinCC 有关的按字母顺序排列的术语。用户可能已经熟悉了这里所解释的大部分术语。
HMI 人机界面
PLC 可编程逻辑控制器
CS 组态系统
RT 运行系统

3 组态 - 常规主题

本节提供了大量关于如何使用 WinCC 来管理项目的信息、指令和方法。其中部分信息可能不只针对 WinCC。
在理想情况下，这些组态规则应成为组态和设计运行系统项目的样式向导。

3.1 在项目开始之前的工作

开始组态前，需要制定大量的规定并进行结构化工作。这样可以

- 简化组态
- 提高项目的透明度
- 简化按组进行工作
- 提高项目的稳定性和性能
- 简化项目的维护

对结构化方针的明确规定是建立或扩展共同标准的基本先决条件。

这些规定可以分为两类：

组态规定

在项目开始前，应定义以下规定：

- WinCC 项目的名称
- 变量的名称
- WinCC 画面的名称
- 创建脚本和动作的规则
- 组态规则(共同标准、库函数、按组工作)
- 归档项目的模式和方法

运行系统项目的规定

有关运行项目的规定(组态结果)。这些规定很大程度上取决于应用领域(例如汽车工业、化学工业、机械制造等)。应该定义下列规定：

- 用户界面(画面安排、字体和字体大小、运行语言、对象的显示等)
- 控制概念(画面体系、控制原理、用户权限、有效键操作等)
- 用于消息、限制值、状态、文本等的颜色
- 通讯模式(连接类型、更新的周期和类型等)
- 数量表(报警、归档值、趋势、客户端程序等的数目)
- 消息和归档方法

3.2 详细规定

在手册的这部分中，将制定用于实例项目的规定。创建用户自己的项目时，这些规定可以作为模板类型使用。

注意：
在我们的实例项目中，项目、画面、变量、变元的名称以及脚本注释都是英语。

组态工具的缺省值

在大多数 WinCC 的编辑器中，可以将某些属性设置为缺省值。因此，WinCC 支持用户组态的特定样式，从而能为指定的任务进行最优化的组态。

注意：
对此的一个实例是可以在图形编辑器工具设置处进行设置的选项。
该主题的详细描述可参阅图形编辑器的在线帮助。

3.2.1 规定：WinCC 项目名称

常规信息

建议将项目名作为文件夹的缺省名称，WinCC 项目中的所有数据都存储在该文件夹中。可以在创建项目初期或以后改变文件夹名称(从 Windows Explorer 中)。

参数/限制

除一些特殊字符(例如\ ? ' . ; : /)之外，允许使用所有的字符。还允许使用数字值 0 - 9。

规定

对于组态手册第二部分中所描述的实例项目，其项目名使用：
a...a_nn
在此：

a	类型标志(a-z、A-Z、没有特殊字符)
_n	序列号，用于区分一些相同类型的项目(编号 0 - 9)，范围为 00 - 99

实例：cours_00.mcp 或 pictu_01.mcp

常规应用的注意事项

例如，WinCC 项目名可以用来区分设备的不同部分。

注意：
更新文档时，可以打印输出 WinCC 项目名。这样使关联和查找信息更容易。

3.2.2 规定：变量名称

常规信息

变量名称可以多于 8 个字符。不过仍应尽量避免太长的名称。如果分配变量名称时严格按照规则进行，则在组态时能更方便的进行查找。
创建 WinCC 项目时，变量管理器的结构对于确保在运行时快速而有效的组态和高性能处理至关重要(在脚本中)。

在定义变量名称之前，需要考虑一些与 WinCC 中变量管理器的结构有关的特殊字符。创建组只能影响组态时变量在变量管理器中的显示方式。组名称将影响变量名称的唯一性。用于 WinCC 项目的变量名称必须是唯一的。系统将检验其唯一性。
WinCC 可以帮助你用不同的方法选择变量，例如通过按列排序(名称、创建日期等)或通过使用过滤器。如果变量名称还包含了其它信息，则用户将会发现这非常有用。

规定

下列规定将应用于本手册中所提及的实例项目的变量名称：
xxxy_z...z_a...a_nn
在此：

x	简写	类型
	BIN	二进制变量
	U08	无符号 8 位数(无符号)
	S08	有符号 8 位数(有符号)
	U16	无符号 16 位数
	S16	有符号 16 位数
	U32	无符号 32 位数
	S32	有符号的 32 位数
	G32	IEEE 754 32 位浮点数
	G64	IEEE 754 64 位浮点数
	T08	文本变量 8 位字符集
	T16	文本变量 16 位字符集
	RAW	原始数据类型
	TER	文本参考
	STU	结构类型

y 简写 原型
- r PLC 的只读变量(读)
- w PLC 的写和读变量(写)
- i 没有链接到 PLC 的 WinCC 内部变量
- x 间接寻址变量(其内容是变量名称的文本变量)

_z 组(对应于现场区域或办公楼)
_Paint ...例如现场区域的名称
_a 变量名称(例如测量点的名称)
_EU0815V10 ...例如测量点的名称
_n 实例的序列号(编号 0 - 9)，范围为 00 - 99

参数/限制

对于变量名的分配，应遵守以下限制条件：

- 特殊字符@应专门保留用于 WinCC 系统变量(然而，在其它地方仍可使用这个字符)。
- 不能使用特殊字符'和?。
- 特殊字符"和字符串//由于在 C 脚本(字符串的开始和结束以及注释的开始)中具有特殊含义，因此不能使用。
- 无空格。
- 变量名称中的字母不区分大小写。

常规应用的注意事项

实例中分配的变量名称仅作为参考。
在脚本和 Excel 中使用变量时，为变量名称的各个部分保持固定的长度是非常有用的(如果必要，使用 0 或 x 作为填补字符)。
例如，可以在 EXCEL 中对大量变量进行有效而简便的创建和维护。如果变量名称有固定结构，则在 EXCEL 中创建变量列表就会相当容易。然后使用在 WinCC 光盘上的程序\SmartTools\CC_TagImportExport\Var_exim.exe 将 Excel 中创建的变量列表导入到当前的 WinCC 项目中。

3.2.3 规定：画面名称

常规信息

如果想要在脚本或外部程序中寻址画面，则使用固定结构的画面名称会非常有用。但是，也需要考虑如何确定画面名称的长度。太长的名称(文件名)不容易识别(列表框中的选择、脚本中的调用等)。根据经验表明，长度最好不超过 40 个字符。

参数/限制

对于画面名称应遵守以下限制条件：

- 最大长度为 255 个字符
- 除某些特殊字符(例如/ " \ : ?
- 画面名称中的字母不区分大小写。

规定

下列规定将应用于本手册中所提及的项目的画面名称：
aaaaa_k_x...x_nn
在此：

a	对画面分组的画面标识符(a-z、A-Z、无特殊字符)。 course...例如 C 教程中的画面名称	
_k	画面类型 0 - 99 的标识符	画面类型
	_0	起始画面
	_1	总览画面
	_2	按钮画面
	_3	设备画面
	_4	详细画面
	_5	消息画面
	_6	趋势画面
	_7	...
	_8	...
	_9	诊断画面(仅用于测试或调试)
_x	画面功能描述的名称(a-z、A-Z、无特殊字符)，最大长度为 30 个字符。 _chapter ...例如 C 教程中章节的名称	
_n	类型的序列号(编号 0 - 9)，范围为 0 - 99	

常规应用的注意事项

实例中的画面名称仅供参考。但是必须遵守用于某些脚本的名称约定。

3.2.4 规定：脚本和动作

常规信息

可以在 WinCC 项目中创建自己的脚本和动作。分配的名称应具有表示性。这样在以后使用脚本时会使事情变得容易得多。
在全局脚本编辑器中组态时，使用比例字体可能会带来麻烦。因此，选择宽度为常数字符的字体(例如 Courier)可能会较容易读取。
应为脚本配上恰当的注释。与需要花费大量的时间在理解注释极为糟糕的程序上相比，编写注释所化费的时间要少得多。但人们往往忽视这个事实。

规定

下列规定将应用于本手册项目中的脚本：
使用比例字体 *Courier New*，大小为 8
所有的变量名称和注释都是英语

常规应用的注意事项

有关如何使用脚本、动作和编辑器的详细描述可以参考 C 脚本的开发环境一章。

3.2.5 规定：用户界面

常规信息

非常仔细地建立用户界面极为重要。已在**图形编辑器**中创建的所有对象将显示在用户工作空间的画面上。
所创建的画面是机器和用户之间的唯一界面。由于其对于确保项目的成功具有重要作用，因此必须特别谨慎。当然设备的操作肯定要比画面的外观重要得多，但长期使用创建草率的画面可能会有不良的影响，甚至增加设备维护的成本，因此需要认真对待。
这些就是操作员(客户)每天都查看的画面。
画面显示系统中，设备当前状态的信息通过显示画面单独呈现给用户。因此界面必须尽可能地提供全面以及容易理解的信息。
WinCC 能够根据用户的要求准确地组态用户界面。根据使用的硬件、处理要求和已有的规定来设置自己特定系统的用户界面。

用户

组态用户界面时，最需要关注的是用户，因为组态毕竟是为用户而执行的。
如果能成功而又清楚地提供给用户所需的信息，那么就会有**较高的产品质量和极少的故障**。同时还能简化必要的维护工作。
用户需要获得尽可能多的信息。使用这些数据作为基础，用户能够作出重要决定

以保持过程以高水平质量运行。用户主要的工作不是对报警作出响应(此过程在这里已经不需要考虑)，而是运用其经验、过程知识和操作系统提供的信息来预知过程发展的方向。用户应该能够在不规则事件发生前进行阻止。WinCC 提供了有效编辑和向用户显示这类信息的可能。

画面内应该包含多少信息?

当确定应集成到画面中的信息量时，为达到平衡，需要权衡以下两方面：

- 如果画面包含的信息太多，读取信息会有困难而且信息的搜索可能会花费较多的时间。用户发生错误的概率也会随之增加。
- 如果画面包含的信息太少，将会增加用户的工作量。用户可能会找不到过程，只能频繁地切换画面以找到所需信息。这样会导致延迟响应、控制输入以及过程控制不稳定。

调查显示，有经验的用户希望**每个画面中包含尽可能多的信息**，从而不需要经常切换画面。

相反，如果一个画面中包含了大量信息，对于初学者而言会感到困惑并不能确定应该怎么做。他们可能不能找到正确的信息或不能及时找到信息。

但是经验告诉我们：初学者不久就会成为有经验的用户，但是有经验的用户不可能再次成为无经验用户。

隐含信息

显示的信息应该重要并容易理解。可以在某段信息(例如测量点标识符)不需要时将其隐含。

显示信息

显示模拟量数值时，可以用数字化的指针仪来表示。数值的图形表达(例如指针仪、棒图...)可以使用户更容易、快捷地识别和掌握信息。

为了避免类似色盲这些不太可能发生的情况，不仅应使用不同的颜色而且还要用不同的格式来表示对象的改变(状态)。

画面中包含的重要信息应总能立即识别出。因此颜色对比度的运用至关重要。

颜色编码

人眼识别颜色的速度要比文本快。使用颜色编码能够非常快速地建立各种对象的当前状态，但是必须建立并始终遵循颜色编码图表。用来显示项目状态的统一的颜色规定(例如红色表示出错/故障)已成为标准。还必须考虑客户已在使用的共同标准。

显示文本

为使文本更易于阅读，应该遵守一些简单的规则。

- 文本的大小不仅必须与文本中所包含信息的重要性相匹配，而且需要考虑用户与屏幕之间的距离。
- 优先使用小写字母。虽然大写字母在较远距离时容易阅读，小写字母要比大写字母少占空间并易于读取。

- 水平文本比垂直或斜置文本易于读取。
- 为不同的信息类型使用不同字体。(例如测量点名称、注释等)。

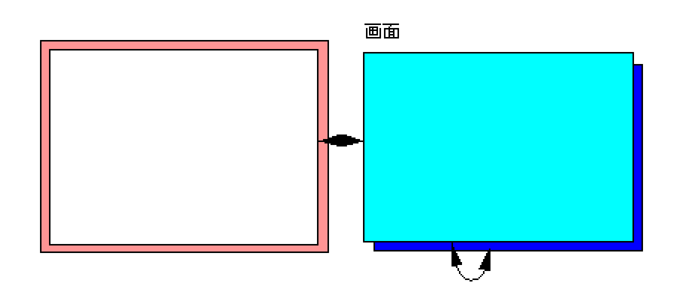
紧扣概念

无论决定使用什么概念，都要坚持让它贯穿整个项目。这样，就可以对过程画面进行直观控制。用户错误就会较少发生。
这也同样适用于所使用的对象。无论是在什么画面中，所绘制的电动机或泵总是一样的。

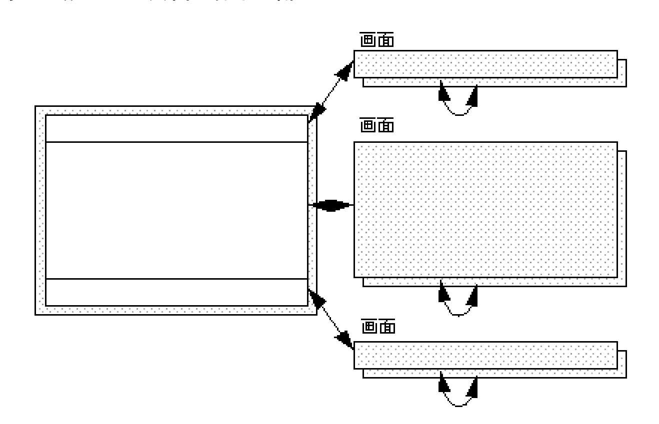
画面布局

如果使用的是标准 PC 监视器，经验表明有必要将画面分为三部分：总览部分、工作空间部分以及按钮部分。
然而，如果在特殊工业 PC 上或具有集成功能键的操作面板上运行应用程序，则对画面内容进行分割的方法并不一定适用。

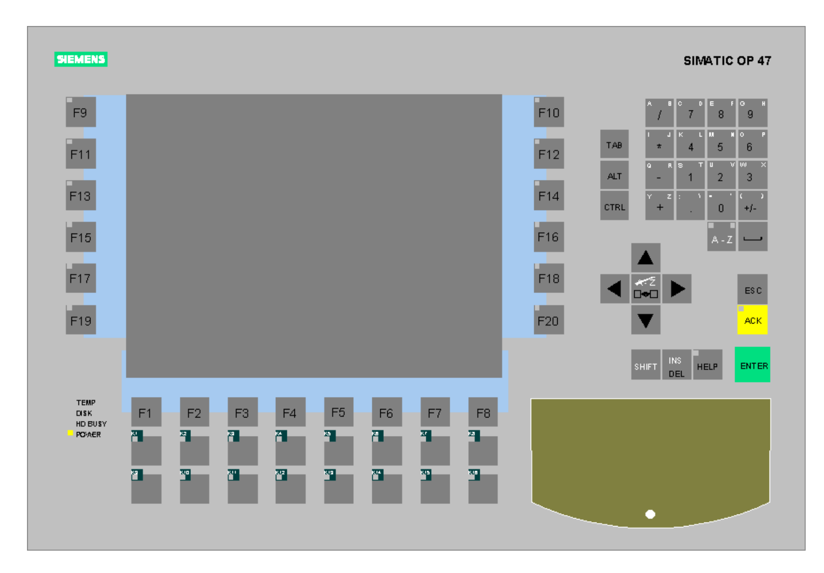
画面占据整个屏幕区域



屏幕分成总览、按钮和设备画面三部分



操作面板实例



参数/限制

各个画面的大小可以根据需要在固定范围内设置(最小 1 x 1，最大 4096 x 4096 像素)。如果单用户系统使用 17"监视器，推荐使用的最大分辨率为 1024 x 768 像素。对于多用户系统(多台 VGA)，较高分辨率会非常有用。
对于操作面板，可用的分辨率通常受到技术的限制(TFT 分辨率从 640 x 480 至 1024 x 768 像素)。

规定

下列规定应用于本手册项目中的画面：

分辨率

在本实例项目中，使用的分辨率为 1024 x 768 像素和 800 x 600 像素(对于特殊情况)。PC 的颜色数目必须设置为最小 65536 色，以使实例项目正确显示。

文本

测量点名称为 Courier 字体，纯描述语句、所有其它的文本和文字显示为 Arial 字体。对于具有视窗风格的消息对话框，使用 MS Sans Serif 和 System 字体。
必要时可调节字体大小。

画面中的信息

不管是否有用，我们在画面中隐含某些信息段。该信息仅在需要时(手动操作或自动)显示。
我们还在项目中使用一些不同的画面布局。如果画面包含大量可控对象，则将以工具提示的形式提供如何使用它们的信息。

画面布局

将对基本选项进行组态来设定画面。然而，在其它项目中，我们将采用将画面分为页眉、工作空间和页脚的方法。

常规应用的注意事项

可以为自己的项目反复使用概念的基本布局。

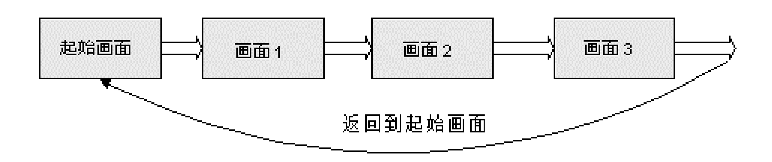
3.2.6 规定：控制概念

常规信息

用户使用常用的输入设备(如键盘、鼠标、触摸屏或工业操作杆)可以在 WinCC 下控制过程应用程序。如果计算机处于极个别不能使用鼠标的工业环境中，则可以组态 tab 顺序和 alpha 光标。通过 tab 顺序在可控制域之间移动，通过 alpha 光标在输入域之间移动。
可以锁定每个控制操作以防止未经授权的访问。

打开画面

选择画面的概念取决于多个因素。最主要的因素是将显示的画面数和过程结构。在小的应用程序中，可将画面设计为循环或 FIFO 缓冲器。



如果运行大量画面，必须设计一个合理的结构体系来打开画面。选择简单而永久的结构以便操作员能够快速了解如何打开画面。
当然，可以直接打开画面，但这对于小的应用程序(例如冷藏库)来说很有意义。

体系

结构体系使过程容易理解、能够简便地处理和提供信息并且快速访问详细信息(如果需要)。
通常且频繁使用的结构体系主要由三个层面组成。

层面 1

归类到层面 1 的是总览画面。
该层面主要包含不同系统部分在系统中所显示的信息，以及如何使这些系统部分协同工作。
该层面还显示在较低层面中是否有事件(消息)发生。

层面 2

归类到层面 2 的是过程画面。
该层面包含指定过程部分的详细信息，并显示哪个设备对象属于该过程部分。
该层面还显示了报警对应的设备对象。

层面 3

归类到层面 3 的是详细画面。
该层面提供各个设备对象的信息，例如控制器、阀、电机等。并显示消息、状态和过程值。如果合适的话，还包含与其它设备对象交互工作有关的信息。

规定

下列规定将应用于在创建本手册的过程中所开发的项目：
将在项目中使用一些不同的控制概念并指出它们的差别。

常规应用的注意事项

手册中的项目仅作为用户创建自己的控制概念的参考。当扩充设备时，必须考虑现有的控制概念。许多用户在进行组态时，都应该遵守公司已制定好的共同约定和标准。

注意：
WinCC 选项包基本过程控制提供了现有的控制概念。
选项包还包含其它有用且强大的功能(例如存储)。

3.2.7 规定：颜色定义

常规信息

颜色是有关 HMI 系统的一个基本讨论主题。WinCC 允许自由地选择线条、边框、背景、阴影和字体的颜色。可以选择 Window 支持的所有颜色。当然也可以在 WinCC 运行中改变颜色以及其它图形元素。
颜色定义对于确保进行简单地组态和清楚地表现过程特别重要。
始终要为下列区域定义颜色。可以根据与 VDE 0199 相关的 DIN EN 60073 来定义颜色，但这需要获得用户同意：

- 消息(激活/清除/确认)的颜色
- 状态(开/关/故障)的颜色
- 字符对象(环形线/填充量)的颜色
- 警告和限制值的颜色

规定

下列规定应用于本手册中项目的颜色：
为了正确显示实例项目，PC 的颜色设置必须大于 256 色。
为了方便查找，将为实例项目中独立的主题(变量、C 教程、画面组态)使用不同的背景色。总览和按钮部分为暗背景色。
对于报警系统，每个消息级别和分配给消息级别的消息类型都分配了某种颜色代码。

常规应用的注意事项

如果需要，应在定义颜色后调整 WinCC 的缺省设置。
*C 动作*内的颜色编码值表可参见附录的颜色表一章。

3.2.8 规定：更新周期

常规信息

当确定更新周期时，始终要从整体上考虑系统：需要考虑更新什么以及更新几次。不恰当的更新周期对 HMI 系统的性能具有反作用。

当着眼于整个系统时(PLC - 通讯 - HMI)，可以在过程中(PLC)检测到改变的发生。在大多数情况下，总线系统是数据传送的瓶颈。

当指定测量值的更新模式时，需要关注测量值实际改变的速度。对于容量为 5000 升的锅炉的温度控制，以 500 ms 的时间间隔进行实际值的更新是毫无意义的。

32 位 HMI 系统

WinCC 是基于 Windows NT 的纯 32 位 HMI 系统。该操作系统已为事件驱动的控制操作进行优化。如果组态 WinCC 时考虑上述因素，即使在处理大量的数据时都不会有性能方面的问题。

规定

下列规定应用于本手册项目的更新：

在任务定义允许的范围内，由事件驱动进行更新。由于我们主要使用内部变量，因此经常通过变量的改变来触发。如果使用外部变量，将根据过程驱动程序连接引起系统负载的增加。如果通讯允许事件驱动的传送，则应选择临界数据。可以由 HMI 在适当的周期内(轮询过程)检索非临界数据。

常规应用的注意事项

有关更新周期的应用的详细描述可参见 更新周期 - 怎样设置和在何处设置一章。

3.2.9 规定：用户权限

常规信息

当操作设备时，需要保护某些操作员功能以防止未经授权的访问。进一步的要求是只有专门的人员才可以访问组态系统。
可以指定用户和用户组，并在用户管理员中定义各种授权等级。这些授权等级可链接到画面中的控制元素。
可以分配基于个人的不同授权等级的用户组和用户。

规定

在实例项目 *Project_C-Course* 和 *Project_TagHandling* 中，每个用户被授权控制项目的操作。
在实例项目 *Project_CreatePicture* 中，用户只能在登录后控制项目的操作。口令与项目名称一样（*Project_CreatePicture*）。
将用于选择单个主题的按钮链接到名为项目控制的授权等级中。

常规应用的注意事项

关于如何分配用户权限的描述可以参见组态手册的第二部分中的实例项目 *Project_CreatePicture*，章节为用户授权。

3.2.10 规定：报警

常规信息

通常，WinCC 支持两个报警步骤：

- 位消息步骤是允许从任何自控系统报告消息的通用步骤。WinCC 监控所选择的二进制变量的信号边沿变化并从中产生消息事件。
- 序列报表要求自控系统本身能够生成消息并以预定义格式向 WinCC 发送可能带有时间标志和过程值的消息。通过消息的操作步骤可以给来自不同自控系统的消息序列排序。参见 S5 报警系统的文档一章。

报告什么？

当指定要报告的事件和组态时，大多数人会根据其认为最安全的方式进行操作，设置软件来报告所有事件和状态改变。这样就要由用户来确定其首先要看的消息。

如果在设备中报告的事件太多，则经验告诉我们将只选择重要的消息以免来不及查看。

常规应用的注意事项

怎样显示消息以及选择哪些消息用于归档，这可根据自己的需要来进行改变和自定义。

3.2.11 规定：关于执行过程

常规信息

执行项目时使用固定的结构来存储数据特别有用。可开始规定 WinCC 项目将在哪个驱动器上进行创建。下一个步骤包括文件夹结构等。

经验表明，最好在包含相关子文件夹的一个文件夹中存储项目的所有数据。此方法不但在处理项目时具有优势，在备份数据时更是如此。

注意：

PC 组态有很大的不同。在为要处理的项目分配驱动器时为了避免由此可能产生的问题，建议使用虚拟驱动器。可以在任何时候改变为虚拟驱动器分配的文件夹。

指定文件夹

除由 WinCC 创建的文件夹外，可根据需要为 Word、Excel 和临时文件创建其它的文件夹。

3.3 用 WinCC 组态时的特性

下列几章主要介绍了用 WinCC 组态的各个方面。
这些主题作为 WinCC 在线帮助的补充。

3.3.1 更新周期-怎样设置和在何处设置

指定更新周期是在可视化系统中执行的最重要的设置步骤之一。设置影响下列属性：

- 画面结构
- 可视化站(图形编辑器)上当前打开的画面中的对象更新
- 后台脚本(*全局脚本*)的处理
- 数据管理器和过程通讯的激活

当测量值根据归档次数处理(变量记录)时设置其它的时间变量。

数据管理器

当前的变量值由数据管理器(变量管理的主要管理器)根据设置的更新周期来请求。参见在 WinCC 中添加动态一章。
数据管理器通过通讯通道获得新过程数据，然后将这些数值提供给应用程序。因此数据请求表示在不同任务(图形编辑器、数据管理器等)之间进行切换。根据组态，这可能会导致完全不同的系统负载。

3.3.1.1　画面中的更新

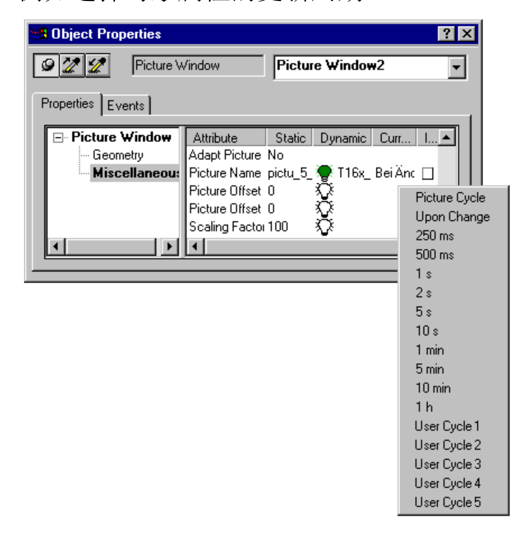
画面更新

更新画面中对象的各个属性指的是打开画面之后已经动态化的对象。更新周期的任务是建立画面中特定对象的当前状态。组态人员或系统可以为下列动态类型设置动态化对象的更新周期：

动态类型	缺省设置	组态自定义
组态对话框	变量触发 2 秒 或 事件触发(例如控件)	自定义时间周期
动态向导	可以根据动态类型从下列选项中选择： • 事件触发 • 时间周期 • 变量触发	自定义时间周期、事件或变量
直接连接	事件触发	
变量连接	变量触发 2 秒	自定义时间周期
动态对话框	变量触发 2 秒	自定义时间周期、变量触发器
关于属性的 C 动作	时间周期 2 秒	自定义时间周期、变量触发器 直接从 PLC 读取
对象属性	设置取决于动态	编辑更新周期列

要选择的更新周期由 WinCC 指定，并且可以由用户定义的时间周期来补充。

例如选择对象属性的更新周期：



3.3.1.2 更新周期类型

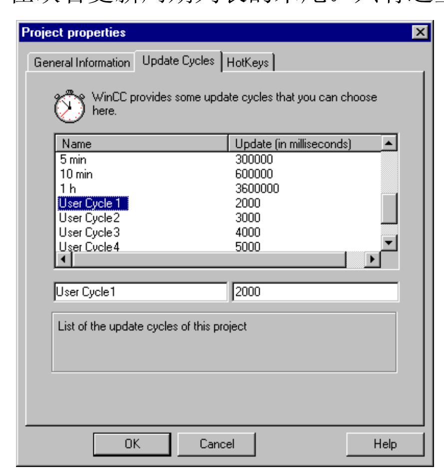
对于更新周期，有以下几种不同类型：

类型	缺省设置
缺省周期	时间周期为 2 秒
时间周期	2 秒
变量触发	2 秒
画面周期	2 秒
窗口周期	一旦改变
用户定义的时间周期	用户周期 1：2 秒 用户周期 2：3 秒 用户周期 3：4 秒 用户周期 4：5 秒 用户周期 5：10 秒

用户周期

至多可以定义 5 个与**项目相关**的用户周期。如果在 *WinCC 资源管理器*左边的树型结构中选择项目名称，则单击如下所示的工具栏按钮将打开*项目属性*对话框。

在*项目属性*对话框的*更新周期*标签内，为项目相关的定义提供的自定义周期 1-5 在缺省更新周期列表的末尾。只有这些用户周期能够参数化。



这些用户周期允许定义除现有时间周期(例如 200 毫秒)之外的其它时间周期。
可以将用户周期定义为 100 毫秒与 10 小时之间的任何时间长度。并且可以为用户周期命名。
这些与项目相关的时间单位可用于所选择的对象，这些对象的更新周期稍后必须进行修改。更改时间周期的原因曾经可能是为了进行优化。用户定义的更新周期也使用户随后可以从单个中央单元修改设置的时间周期。在这种情况下，不必再调整画面的各个对象。如果要使项目易于维护，那么就应该优先考虑这种定义用户周期的方法。

3.3.1.3 更新周期的含义

在开始使用更新周期之前，首先必须考虑各种更新周期的含义。
对于更新周期，有以下几种不同的类型：

类型	含义
缺省周期	时间周期
时间周期	根据设置的时间，各个对象的属性或动作将会更新。也就是说数据管理器**逐个**请求变量。
变量触发	按照设置的周期时间并且经过时间间隔之后，系统确定变量并且检查其数值的变化。 如果在设置的时帧期间所选变量至少有一个其值发生变化，则这作为与此相关的属性或动作的触发。 所有**变量值由数据管理器**一起**请求**。
画面周期	更新当前画面对象和通过画面周期的更新周期触发的所有对象的属性。
窗口周期	更新窗口对象和通过窗口周期的更新周期触发的所有对象的属性。
自定义的时间周期	可以专门为一个项目定义的时间单位。
直接从 PLC 读取的 C 动作	通过 C 动作中的内部函数，可以从 PLC 直接读取数值。C 动作中后继指令的进一步编辑只有在读取过程值(同步读取)之后才能继续进行。

注意：

在每一种情况下，由数据管理器请求的当前变量值会导致任务的改变以及各任务之间的数据交换。此外，数据管理器必须通过相连的可编程控制器的通讯通道请求变量值。根据通讯类型，它通过发送到通讯接口(FETCH)的请求报文和从可编程控制器返还至 WinCC 的数据报文来完成。

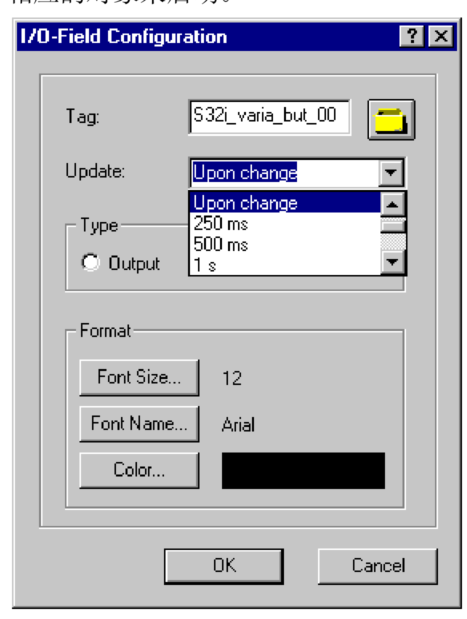
3.3.1.4 有关更新周期应用的信息

对于更新周期的应用，根据所选的周期类型推荐使用下列设置：
以下实例显示设置每种更新周期的位置：

类型	缺省设置	推荐的组态
缺省周期	时间周期为 2 秒	*动态对话框或 C 动作*： 如果变量是互相影响的,应该在所有的事件中使用变量触发。这样可以减少任务数量的变化和任务间的通讯。 变量触发一旦改变只能有选择地使用，因为它会引起较大的系统负载！变量的变化情况会不断接受检查。这种循环检测机制总是会导致较大的系统负载。 建议标准对象使用 1 至 2 秒的周期。
时间周期	2 秒	根据对象类型或对象属性设置时间周期。过程组件(与开关操作不同的罐填充或温度组件)的惯性同样应该考虑。 建议标准对象使用 1 至 2 秒的周期。
变量触发	2 秒 (*对于动态对话框*)	如果此更新选项可以组态(根据动态类型)，则应该优先使用它！如果变量是互相影响的，则始终要考虑负责更改属性或执行动作的所有变量。只有那些包含在列表中的变量才能作为更新动态属性或动作的触发器。 变量触发 *一旦改变*只应有选择地使用。一旦所选变量中有一个发生变化，就会触发该属性或动作的触发器。这种循环检测机制会导致较大的系统负载。
画面周期	2 秒	只有当画面对象本身的动态属性在较短的时间间隔内发生变化并因此而必须进行更新时，才应该缩短此周期。延长画面周期会减少系统负载。
窗口周期	一旦改变	如果正在处理一个打开的画面窗口，则此设置对调整过程变量(过程框)会起作用。 如果为了获得信息(例如画面布局)而不断显示画面窗口，则应该将窗口及其内容的更新设置为变量触发或时间周期。
直接从 PLC 读取的 *C 动作*		用于同步读取过程值(直接从 PLC)的内部函数(例如 GetTagWordWait)只应有选择地使用。应用这些函数需要由系统进行循环检测(脚本控制)，因此会导致较大的通讯负载。

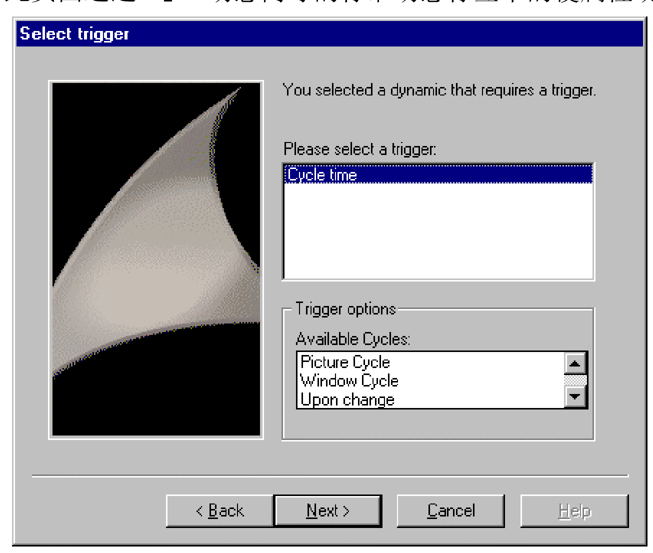
组态对话框

如果组态了*智能对象* → *I/O 域*，则会显示此对话框。此对话框也可以通过 🖱R 相应的对象来启动。



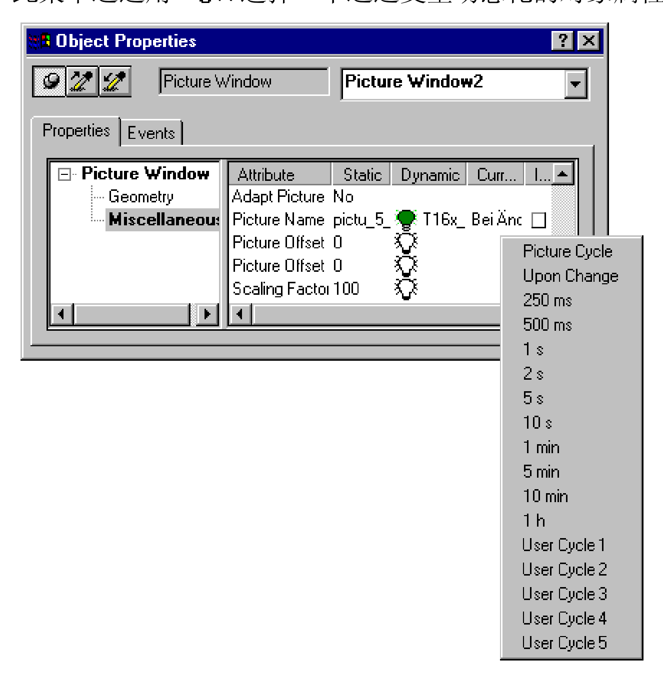
动态向导

此页面通过 ☝D 动态向导的标准动态标签中的使属性动态化条目来显示。



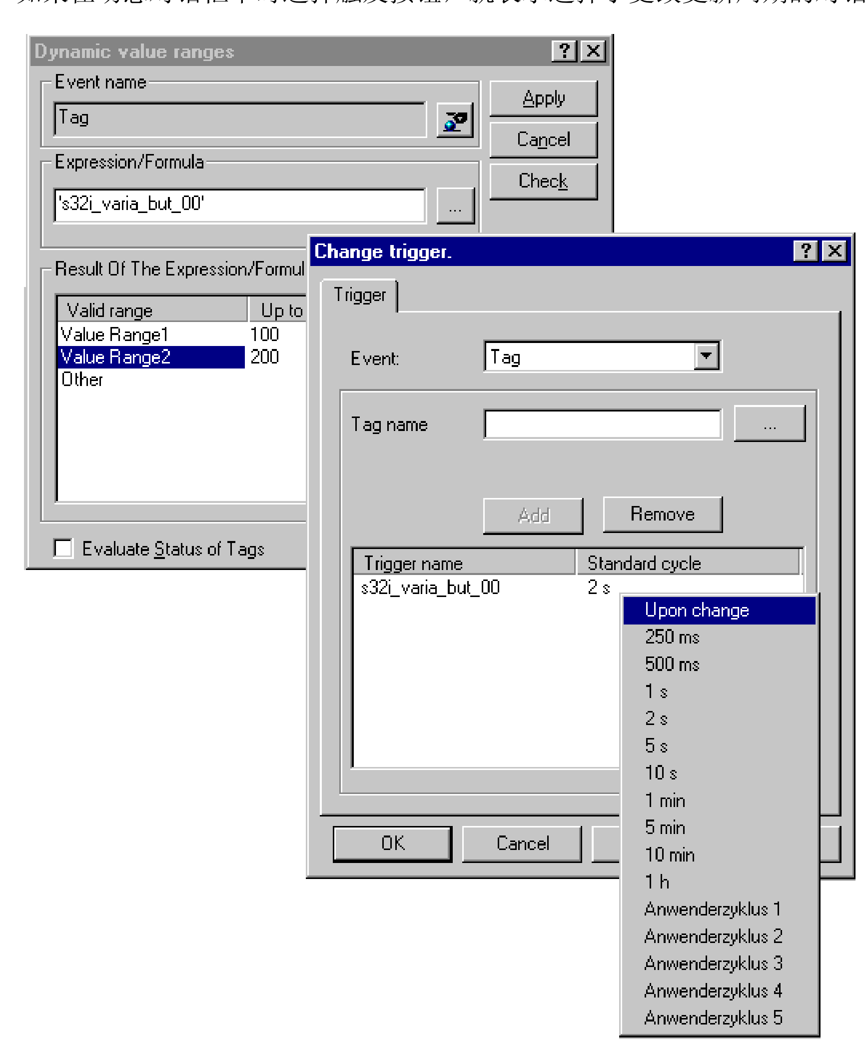
对象属性的变量连接

此菜单通过用 ☝R 选择一个通过变量动态化的对象属性的当前列来显示。



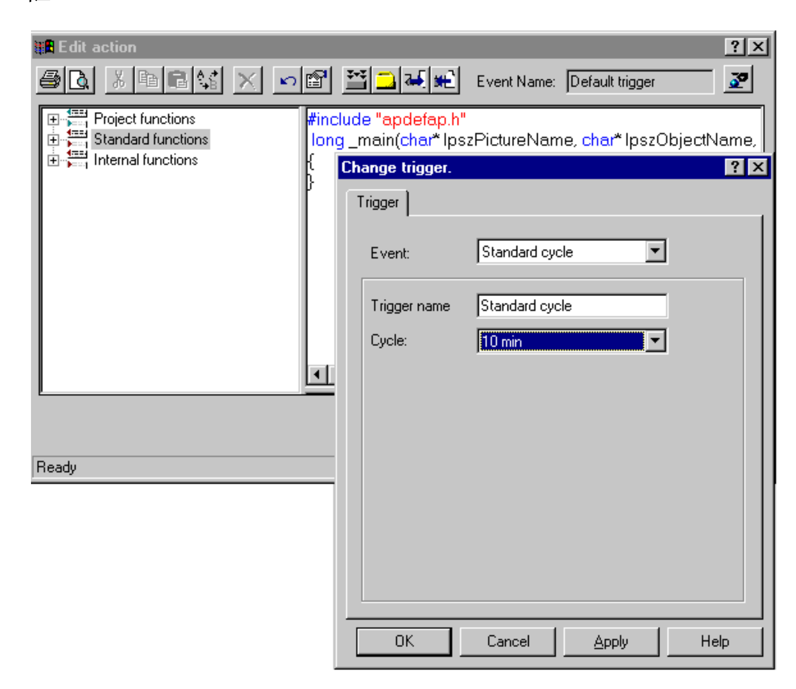
动态对话框

如果在动态对话框中时选择触发按钮，就表示选择了更改更新周期的对话框。



属性的 C 动作

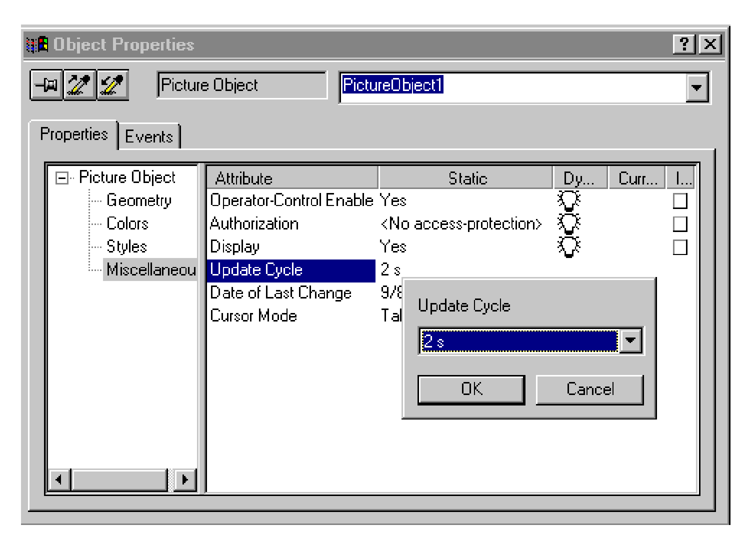
如果组态 C 动作时选择编辑器中的触发按钮，就表示选择了更改更新周期的对话框。



设置的缺省更新周期可以按照如下进行更改：

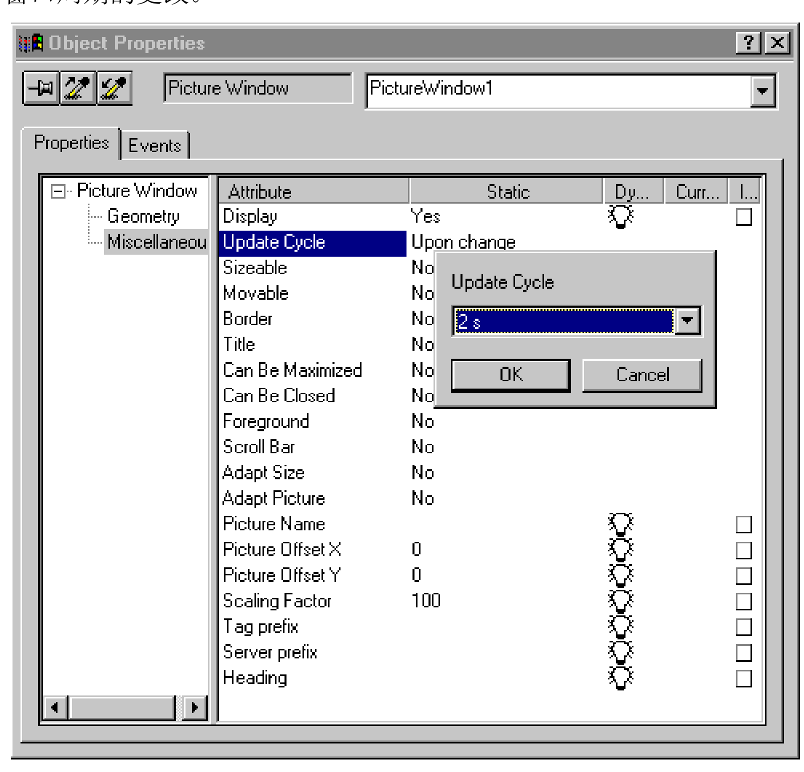
画面周期

画面周期的更改。



窗口周期

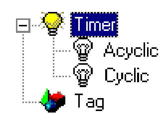
窗口周期的更改。



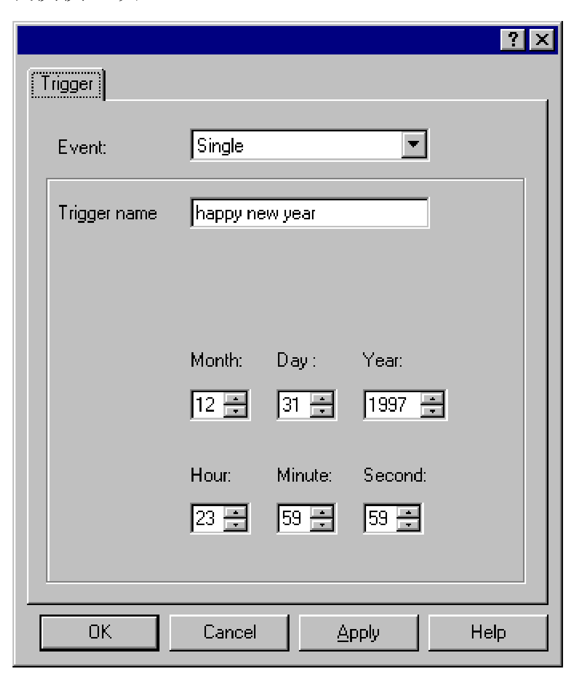
3.3.1.5 后台脚本(全局脚本)的执行

- 后台脚本(全局脚本)的执行取决于组态的各种变量：
- 时间触发(周期性执行，例外：非周期=一次)
- 时间周期
- 时间

事件触发



或仅仅一次



时间周期

已组态的全局动作的时间因子定义何时处理定义的序列动作。除了已描述的缺省周期和相关的 250 毫秒至 1 小时的时间设置(或用户定义的周期 1 - 5)之外，还有其它时间触发：

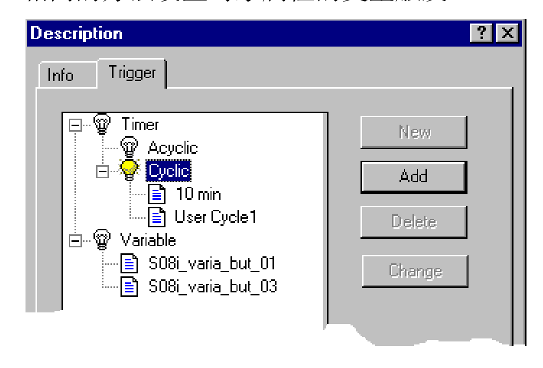
- 每小时 (分和秒)
- 每天 (时、分、秒)
- 每周 (星期、时、分、秒)
- 每月 (日、时、分、秒)
- 每年 (月、日、时、分、秒)

可以选择。

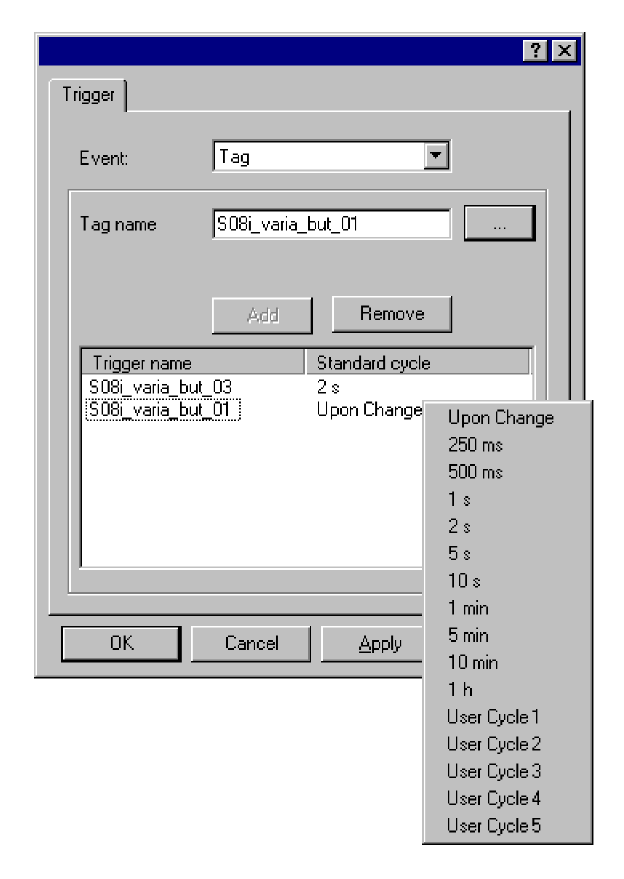


变量触发

如果动作用一个或多个变量来激活，则必须将事件触发器设置为变量触发。按照相同的方法设置对象属性的变量触发。



缺省设置的周期为 2 秒。然而，组态人员可以设置下列时间因子来代替缺省值：



在设置的时帧的开始和结束，确定所选变量的值。如果变量中至少有一个其数值发生了变化，就会触发全局动作。

注意触发一旦改变时的高系统负载。该设置并非始终合适。在此应用的同样地建议适用于更新对象。

定义为全局动作的所有动作不检查和激活对象链接，也就是说只取决于时间周期或事件触发设置。因此，为了不使系统负载太重，只是有选择地使用全局动作并且避免不必要的动作步骤。不要使用太多的时间周期也不要使用太多的短的时间周期来执行动作。

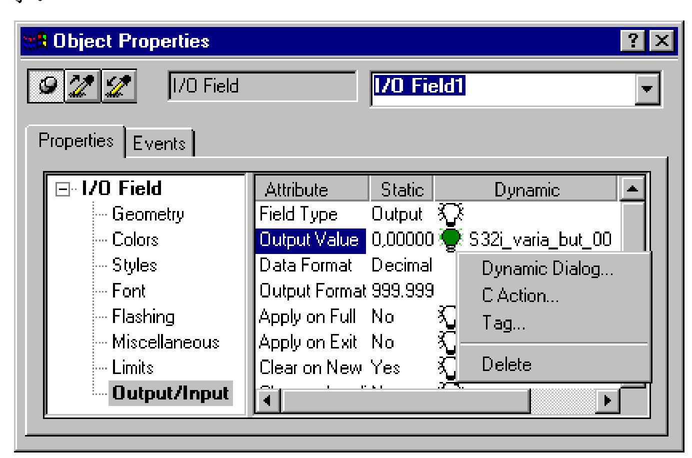
3.3.2 在 WinCC 中添加动态

定义

术语“添加动态”表示运行期间状态(例如位置、颜色和文本等)的变化以及对事件(例如鼠标单击、键盘操作和数值变化等)的响应。
将图形窗口中的每个元素看作独立的对象。图形窗口本身同样是一种称为*画面对象*的对象类型的对象。
*WinCC 图形系统*中的每个对象都具有*属性*和*事件*。除少数情况外，大部分*属性*和*事件*都能够动态化。少数例外情况主要是指运行期间不受影响的属性和事件。它们没有显示其可以动态化的符号。

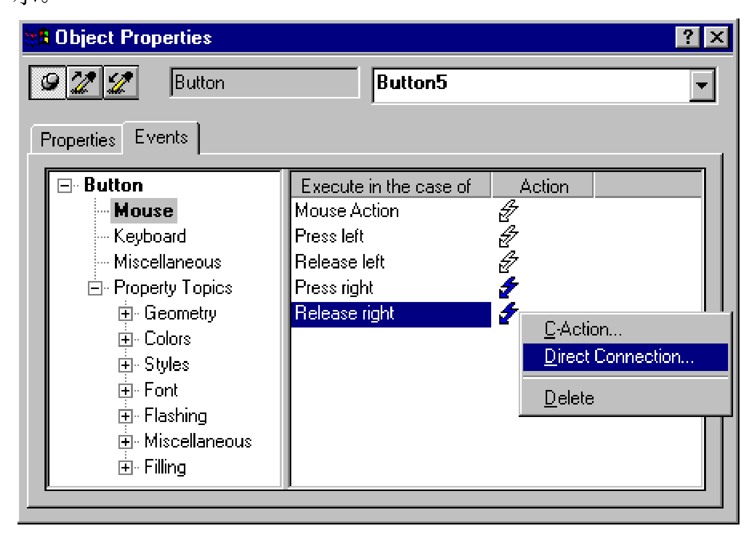
3.3.2.1 使属性动态化

对象的属性(位置、颜色和文本等)可以静态地设置，并且可以在运行期间动态地改变。
动态列中所有带有灯泡的属性都可以动态化。一旦属性被动态化，取决于动态类型的彩色符号就代替白灯泡显示。已经动态化的主题(例如几何结构)以粗体字显示。



3.3.2.2 使事件动态化

对象的事件(例如鼠标单击、键盘操作和数值变化等)可以在运行期间检索，并且可以动态地进行判断。
动作列中所有带闪电符号的事件都可以动态化。一旦事件被动态化，取决于动态类型的彩色闪电就代替白色闪电显示。已经动态化的主题(例如其它)以粗体字显示。



3.3.2.3 对象的动态化类型

设备画面的对象可以用许多不同的方法来动态化。执行动态化的独立标准对话框面向不同的目标区域，并且在某种程度上会导致不同的结果。

总览

动态类型	A	B	优点	缺点
动态向导	x	x	组态时标准的提示方法	只适用于某些动态化的类型。始终生成一个 C 动作！
直接连接		x	**使画面动态化的最快方法，在运行系统中性能最佳**	**只限于一个连接，并且只能在画面中使用。**
变量连接	x		易于组态	对动态化有所限制的选项
动态对话框	x		快速且清楚；用于数值范围或许多的选择项；在运行系统中性能较高	并不适用于所有动态化类型。
C 动作	x	x	由于脚本语言(ANSI-C)非常强大，所以对动态化几乎没有限制。	错误的 C 指令可能会导致产生错误与其它动态化类型相比性能较慢，因此始终要检查是否能通过使用不同的动态化类型来达到目标。

图形说明：

A： 对象属性的动态化
B： 对象事件的动态化

打开进行动态化的对话框

对话框	打开命令
组态对话框	并不是所有对象都有这样的对话框。 自动创建这些对象。 在画面中选择对象 → 按住 SHIFT 键 → D 对象。在画面中选择对象 → R 对象来打开其弹出式菜单 → 选择组态对话框
动态向导	在画面中选择对象 → 选择属性或事件 → 选择向导D 向导来启动。向导必须在查看 → 工具栏....中采用复选标记
直接连接	在画面中选择对象 → 显示对象属性 → 选择事件标签 → R 动作列来打开弹出式菜单 → 选择直接连接。
变量连接	在画面中选择对象 → 显示对象属性 → 选择属性标签 → R 动态列来打开弹出式菜单 → 选择变量 → 在随后的对话框中，选择并应用相应的变量。
动态对话框	在画面中选择对象 → 显示对象属性 → 选择属性标签 → R 动态列来打开弹出式菜单 → 选择动态对话框 → 在随后的对话框中，组态并应用相应的动态
C 动作	在画面中选择对象 → 显示对象属性 → 选择属性或事件标签 → R 动态或动作列来打开弹出式菜单 → 选择 C 动作 → 组态并编译相应的 C 动作。

结果和表现形式

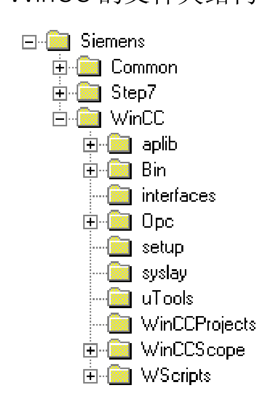
对话框	结果	表现形式（符号）
动态向导	总是生成一个 C 动作。	绿色闪电
直接连接		蓝色闪电
变量连接		绿色灯泡
动态对话框	自动生成 C 动作(InProc)，此 C 动作随后可以扩展，但是会在过程中丢失性能优势。	红色闪电 C 动作一旦改变，就切换到绿色闪电
C 动作	已组态的 C 脚本	绿色闪电 黄色闪电，表示动作仍然要编译

3.3.3 WinCC 系统环境

缺省情况下，WinCC 安装在 C:\Siemens\WinCC\下的文件夹中。在安装期间可以更改此缺省路径。

3.3.3.1 WinCC 系统的文件夹结构

WinCC 的文件夹结构（不带选项和实例）如下所示：



WinCC 缺省文件夹中的文件

在缺省的 WinCC 路径下，下列文件夹和文件对于组态人员和调试工程师非常重要：

文件夹	文件名、扩展名	注释
诊断	License.log	关于许可证检查和/或违约的当前记录条目。
	License.bak	含有最后一次启动的许可证信息的日志文件。
	WinCC_Op_01.log	运行期间由 WinCC 生成的操作员消息。
	WinCC_Sstart_01.log	启动时由 WinCC 生成的系统消息。故障检测时的重要文件。此文件包含有关丢失的变量和错误执行的脚本的消息。
	WinCC_Sys_01.log	运行期间由 WinCC 生成的系统消息。故障检测时的重要文件。此文件包含有关丢失的变量和错误执行的脚本的消息。
	S7chn01.log	所用通道的系统消息(如果是 S7)
aplib	库路径	头文件、所有标准函数和所有内部函数存储在子文件夹中。

WinCC 缺省文件夹中的文件

在缺省的 WinCC 路径下，项目范围的函数和符号存储在下列文件夹下：

文件夹	子文件夹、文件名	注释
aplib	library.pxl	WinCC 缺省库的符号。
	Report、Wincc、Windows	标准函数的文件夹；它们随时可以进行调整。
	Allocate、C_bib、Graphics、Tag	内部函数的文件夹；它们**不能**进行调整。
syslay		创建项目时，由 WinCC 自动将所有打印布局复制到项目路径下的 prt 文件夹中。
wscripts	Dynwiz.cwd	图形编辑器的动态向导。用户可以随时创建自己的脚本。这些脚本的扩展名为.wnf。
	wscripts.deu	此路径包含德语脚本文件。它取决于所安装的语言。
	Wscripts.enu	此路径包含英语脚本文件。由于英语是缺省语言，所以此路径总是会被创建的。
	Wscripts.fra	此路径包含法语脚本文件。它取决于所安装的语言。

缺省 WinCC 文件夹中的文件

安装 WinCC 时，下列应用程序存储在以下文件夹中：

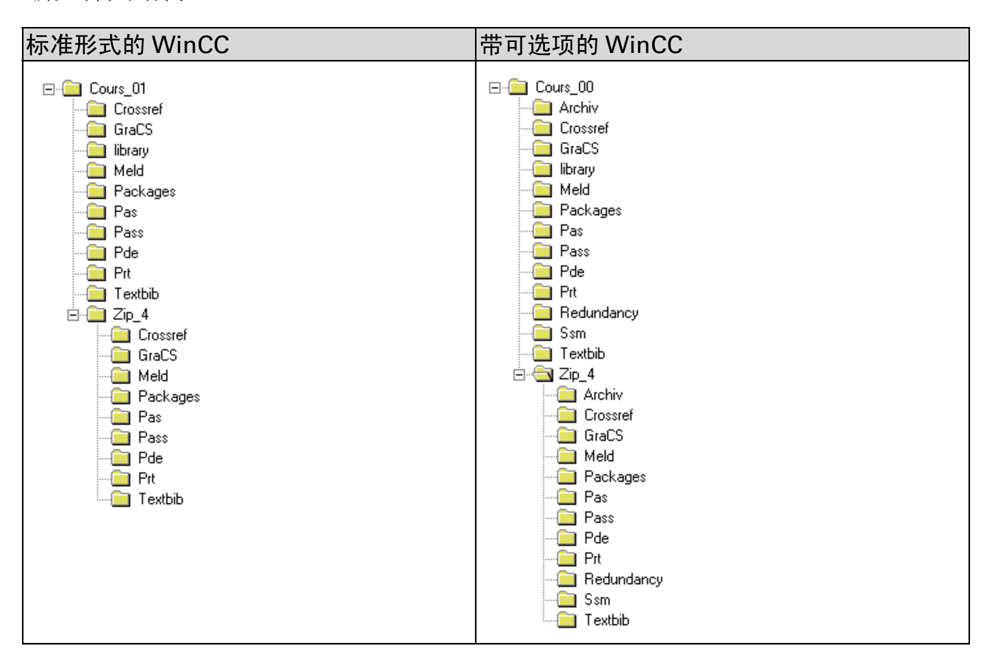
文件夹\文件	注释
\sqlany\isql.exe	用于查看 WinCC 项目数据库中数据的交互式程序。
\bin\Wunload.exe	W 向导用于清空 WinCC 项目数据库中的在线表格，例如删除存储的消息和测量值数据。 向导自动为卸载提供运行表；用户可以随时从列表中添加或删除附加表。 此工具必须在 WinCC 项目的离线状态下使用。它不能在运行系统模式下使用。通过可选的 STORAGE 程序包，可以在运行期间导出消息和测量值。
\bin\Wrebuild.exe	重构数据库的向导；它不能在运行模式下使用！
\SmartTools\CC_GraficTools\metaVw.exe	EMF 格式(扩展图元文件)图形文件(例如打印作业、导出的符号)的浏览器。
\SmartTools\CC_GraficTools\wmfdcode.exe	WMF 格式(窗口图元文件)图形文件的浏览器。
\SmartTools\CC_OCX_REG\ocxreg.exe	用于注册或取消注册附加的 OLE 控制组件(OCX)。
\SmartTools\CC_OCX_REG\Regsvr32.exe	由 ocxreg.exe 调用。

3.3.4 WinCC 项目环境

注意：
为 WinCC 项目创建专用的文件夹，例如 WinCC_Projects。这样就可以使 WinCC 系统和组态的数据彼此完全独立，简化数据备份的任务。如果要卸载 WinCC，也可以避免数据丢失(由于操作员出错)的危险。

3.3.4.1 WinCC 项目的文件夹结构

WinCC 中的项目包括一个具有相应目录的完整的文件夹结构。在 WinCC 资源管理器中创建一个新项目(通过菜单条目文件'新建)之后，就会建立一个如下所示的新文件夹结构：



项目文件夹的内容

文件夹	扩展名	注释
项目路径	.db	含组态数据的数据库。
	rt.db	含运行数据、测量值和消息的数据库。
	.mcp (主控制程序)	WinCC 项目的主文件。项目用此文件来打开。
	.pin	Project.pin
GraCS	.pdl (画面设计语言)	组态的画面。
	.sav	具有最后组态状态的画面文件的备份文件。
	.gif (位图)， .wmf (窗口图元文件)， .emf (扩展图元文件)	画面文件
	.act (动作)	导出的 **C 动作**
	.pdd	Default.pdd 图形编辑器的设置参数(对象选项板中各个对象的缺省设置)
Library	.h (头文件)	**Ap_pbib.h** (项目函数声明)
	.pxl	**Library.pxl** (项目符号库)
	.fct	项目函数
	.dll (动态链接库)	在 C 开发环境下创建的独立函数库。
Meld		
Pas	.pas (动作定义)	根据设置的触发作为后台动作运行的项目动作。
Pass		
Pde		
Prt	.rpl (报表画面语言) .rp1 (行布局)	打印作业的页面布局。 所有预定义的 WinCC 缺省布局都以@开头。所有系统变量都通过此前缀来识别。
计算机名称，例如 Zip-ws1	\GraCS\GraCS.ini	图形编辑器的初始化文件。

可选：组态期间可以创建的文件

文件夹	扩展名	注释
在某种程度上可自由定义	.ini	具有调用信息的模拟器的初始化文件。
	.sim	具有模拟设置的内部变量。
	.csv	从文本库导出的文本。
	.txt	从消息系统(报警记录)导出的消息。
	.emf	将打印结果写入文件的打印作业。
	.log	日志文件
	.xls .doc .wri	用其它应用程序创建的但在 WinCC 项目中使用的文件。

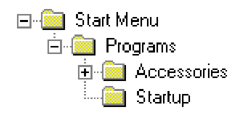
3.3.5 WinCC 中项目的自动启动

要求

当 Windows 启动时，设备中的 HMI 系统(WinCC)应自动启动。HMI 站操作员不必了解任何有关 Windows 使用的事项(例如在 Windows NT 下激活 WinCC)。

解决方案

在 PC 启动例行程序期间，WinCC 自动启动。该功能在 Windows 的启动文件夹下设置。



创建连接

步骤	NT 4.0 中的步骤：
1	在 Windows 资源管理器中，进入 *WinNT\Profiles\All Users\Start Menu\Programs\Startup* 文件夹。WinNT 是安装 Windows NT 的文件夹。
2	在该文件夹中，通过 R → *新建* → *连接*来创建一个新连接。
3	在 WinCC 的*\bin* 文件夹中建立与程序 *mcp.exe* (主控制程序)的连接。
4	为该连接命名。

从而，WinCC 将自动启动。WinCC 本身与最后处理或激活的项目一起自动启动。
为了在运行系统模式下启动系统，项目必须在激活模式下退出。

注意：
如果组合键 *CTRL + SHIFT* 没有锁住并且在 WinCC 启动期间已被按下，则即使项目在激活模式下退出，WinCC 也会在组态模式下启动。

现在操作员可看到熟悉的系统启动画面。为了防止操作员无意或有意地切换至组态(在后台运行)或 Windows 应用程序，必须采取相应的措施。由于操作员可以从数据库运行系统窗口中关闭 WinCC 数据库链接，所以必须使他们不能打开该窗口。

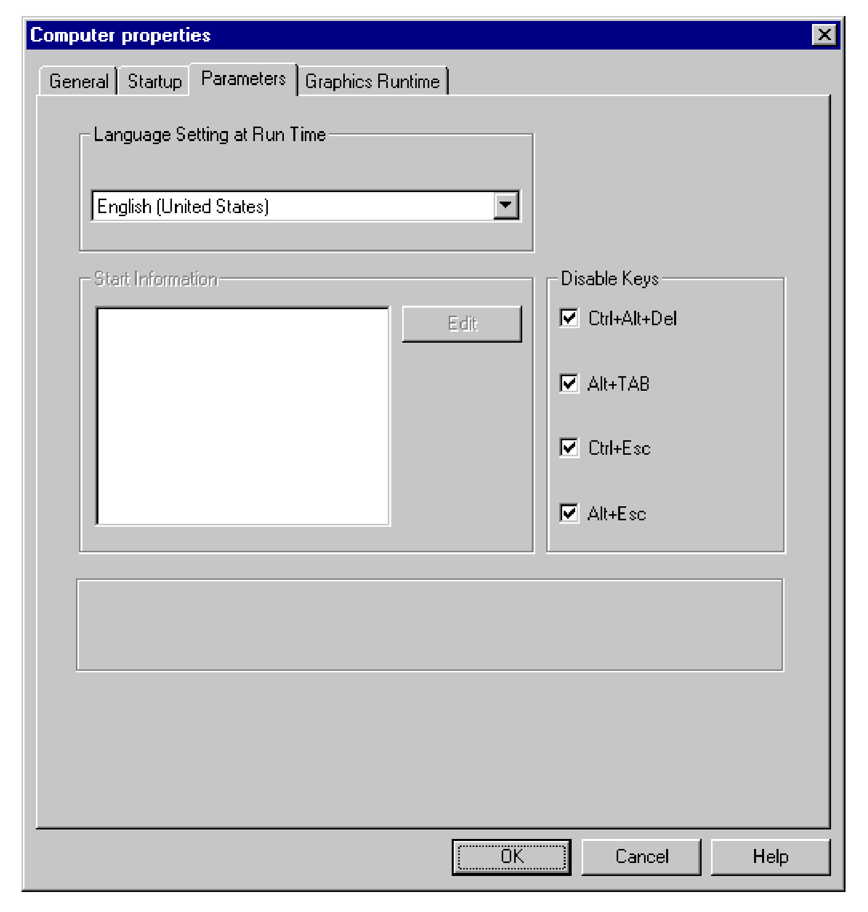
不要选择：

操作员不能从 WinCC 运行系统切换至：

- WinCC 的 *WinCC 资源管理器*(组态环境)
- WinCC 的 SQL 数据库运行窗口(Sybase SQL Anywhere)，因为他们可以使用这种方法来终止 WinCC 数据库连接。这样 WinCC 就不能再运行
- Windows 的任务栏，因为它可以用来启动所有已安装的程序
- 当前的任务窗口，因为从该窗口中可以关闭应用程序

计算机上所需的设置

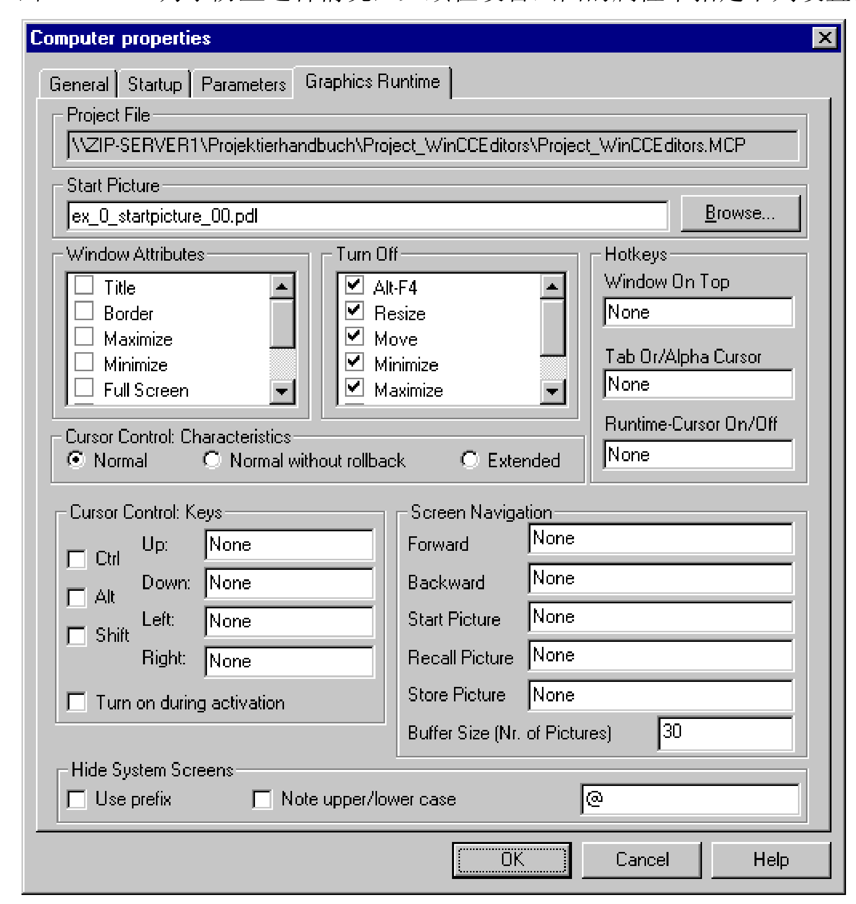
为了防止操作员执行这些操作，必须锁住下列组合键。



在 WinCC 资源管理器中的计算机属性对话框内锁住组合键。
各组合键的确切定义可以从 WinCC 帮助或操作系统帮助中获得。

运行系统所需的设置

通过标准的 Windows 键也能够关闭设备画面；也就是说也可以通过这种方法退出 WinCC。为了防止这种情况，必须在设备画面的属性中指定下列设置：



在 *WinCC 资源管理器*中的*计算机属性*对话框内设置这些锁定。
各组合键的确切定义可以从 WinCC 帮助或操作系统帮助中获得。

- 如果*调整大小*和*最小化*未释放，则可以访问操作系统的用户界面。
- 关闭(上图中未显示)也必须释放，否则用户可以退出运行系统并访问组态系统。

注意：
如果上述键被全部或部分锁住，则必须为组态人员或服务人员提供一个专门为此组态的键，以便访问组态。这也适用于系统的正确关闭。

这些功能不应该很容易就可以访问。通过设置口令属性，也为按钮添加访问保护。

3.3.6 协调关闭 WinCC

保持系统的一致性

为了保持系统的一致性，正确退出 WinCC 是非常重要的。WinCC 运行系统始终应该用预定的按钮先退出。然后，在 WinCC 资源管理器中关闭项目。
如果没有遵守此步骤，则可能会丢失数据、破坏数据库或者导致系统崩溃(最坏的情况)。

受控退出

未关闭操作系统之前，决不要关闭 WinCC 站。使用紧急关机开关不适合 HMI 系统。因此，必须组态一个相应的按钮来允许操作员在不需要任何附加知识的情况下正确退出系统。

电源故障

为了避免由于脉动电流或电源故障而丢失数据，应设计出适合 HMI 系统的详细的数据备份原理并付诸实施。
无论如何都应该为 WinCC 站配置 UPS (不间断电源)。这可以通过将站连接到设备本身的 UPS 上或将独立的 UPS 连接到 WinCC 的服务器上来实现。这对单用户和多用户系统都适用，而与使用的操作系统(Windows NT)无关。
使用的 UPS 必须也包含用于 Windows NT 的特殊关机软件，如果发生电源故障或者在指定的时间间隔后关机，则它可以在不丢失数据的情况下自动退出操作系统和所有激活的应用程序；例如带有用于 Windows NT 的电源关机软件的 APC UPS 600。

3.3.6.1 安装 UPS 的注意事项

WinCC 站必须具有可以连接 UPS (带有相应测试软件)的串行接口。如果站上没有空余的串行接口(例如全被打印机或 PLC 占用了)，则必须另外安装一块接口卡。
由于需要不断对系统进行监控，因此大多数 UPS 系统不支持当然也不建议使用可连接两台或多台外围设备(例如通过开关)的串行接口。
在操作系统中安装合适的监控服务程序。然后将关机参数分配给监控服务程序，以保证协调有序的退出系统。为了在关机时没有任何数据损失地关闭 WinCC，必须在所有情况下都激活应用软件的关机过程。关机前必须选择保存时间，可使所有激活的应用程序有足够时间正确退出。
大多数 UPS 系统软件包还提供了具有时控的关机，例如周末或夜间。该功能也可以用来实现**在没有任何操作员的输入时关闭 WinCC 系统**。

3.3.7 数据备份

何时应备份数据？

必须重复保存和备份 WinCC 项目的原因如下：

- 在组态阶段。
- 在导出或导入数据前(例如在导入变量、多语种画面文本、消息文本和多语种消息文本时)。
- 在建立或卸载 WinCC 数据库前。
- 在使用诸如交互式 SQL access 工具编辑数据库前。
- 必须备份组态数据以在最终用户端的目标系统上进行安装。
- 在沿用类似结构的项目数据前。

什么介质合适？

介质	优点	缺点
软盘	几乎所有的系统都能读取。	容量不够(即使数据进行了压缩)。
ZIP 磁盘	价格适中，容量足够，能通过 Windows 直接而快速访问；便于安装；可携带，能方便地使用在设备中。	
磁带(例如在网络上)	可以自动备份(每日)，容量大。	大多数用于办公室环境，由于使用特殊格式保存，因而不能直接访问数据。
其它 PC 上的硬盘(例如 Laplink)	无需用到其它介质，可以直接使用数据。	速度慢，不适合大量数据。
MOD	高级别的数据完整性，可重复使用，可以在运行方式下备份消息和测量值。	需要使用特殊的驱动器来进行读和写。
CD-ROM	容量大，几乎所有系统都能读取，适合长期归档。	需要使用特殊的驱动器来写入，介质不可重复使用。

在数据备份前简化项目

在为最终用户备份数据或传送项目数据之前，为了简化项目并使其尽量小，可以使用其它程序来删除或简化下列数据。

- 在项目文件夹*GraCS*\ **.sav* 中的所有备份文件。

如果没有为文档创建自己的任何布局(报表编辑器)，则可从\Prt 文件夹删除系统布局。创建新项目时，所有的系统布局将自动从\Siemens\WinCC\syslay 文件夹复制到项目文件夹。

哪些数据必须备份？

如果只要备份 WinCC 项目数据，则必须备份下列各个文件和包含了所有文件的文件夹。

- 从项目文件夹中：
- 文件 **.mcp*、**.pin*、**.db*
- 文件夹*GraCS* 和*Library*
- 如果已创建了自己的动作，文件夹*Pas*
- 如果已创建了自己的打印布局，文件夹*Prt*

如果已创建整个项目的组件(标准函数、项目库中的对象)，则下列文件必须从

- 缺省的 WinCC 文件夹进行备份：
- 文件*Siemens**WinCC**aplib*\ **.fct*
- 文件*Siemens**WinCC**aplib**library.pxl*

重新安装 WinCC 时不产生此数据。

3.3.8 将已备份的 WinCC 项目复制到新的目标计算机

安装系统软件

可以在集成系统上通过自动调用的组态对话框或随 WinCC 程序包提供的安装光盘及授权盘来安装 WinCC 软件。

附件

如果用户项目还要使用其它程序包(例如选项包或附加件)、特殊的通讯接口或其它 Windows 程序(例如 WORD、EXCEL 等)接口，则必须在目标计算机上安装这些程序包。而且在目标计算机上还必须安装选项包、附加件或通讯接口(通道 DLL)的授权文件。注意必须在新的计算机上导入所有需要的授权文件(对于所有使用的 DLL 通道)，以便对 WinCC 项目进行操作。

Windows 软件

如果在 WinCC 画面中使用 OLE 链接到其它 Windows 程序(例如 WORD、ClipArts 或 EXCEL)，则根据不同的 OLE 链接类型同样需要在目标计算机上安装相应的程序包，即输入到 Windows 注册表中。

OCX，ActiveX

如果要使用第三方程序包中的其它 OCX 组件(OLE Control、ActiveX)，必须将它们存储在 Windows 注册表中。注册或检查这些 OCX 组件的注册条目，例如使用 WinCC 光盘提供的工具 SmartTools\CC_OCX_REG\ocxreg.exe。如果在运行模式下(组态期间同样使用图形编辑器)启动 WinCC 项目时没有找到 OLE 对象或 OCX 对象的注册信息，则在画面中显示的该对象被注释为未知对象。

网络

如果项目已为**多用户系统**组态，在复制 WinCC 数据前需要在 WinCC 目标计算机上对网络进行完全安装。在已组态的计算机环境中记录必需的计算机名称，在分配复制项目的参数时会用到这个名称。在单用户系统中还需要使用计算机名称作为参数。因此，必须知道目标计算机的名称或通过 Windows 控制面板来确定其名称。

复制数据和启动项目

步骤	传送数据的过程
1	创建项目文件夹(例如 *WinCC_Projects*)。
2	将整个备份路径作为子文件夹复制到该项目文件夹(例如*WinCC_Projects**Varia_00*)。如果需要，可以改变 WinCC 项目文件夹的名称。在项目文件夹中重命名文件时(varia_00.mcp、varia_00.db、varia_00.pin、varia_00.log)，需要确保给出的所有文件具有相同的名称(扩展名除外)。
3	打开 WinCC 资源管理器中的项目。
4	如果需要，可以改变指定项目的设置。如果要改变类型，在计算机属性中还需进行其它调整。这些调整在 WinCC 帮助中进行了描述。 
5	在常规信息标签的计算机属性处检查计算机名称，如果需要还可进行修改。如果 WinCC 项目中的计算机名称和目标系统中的计算机名称不匹配，则当激活项目时，也就是激活运行系统时，将显示错误消息。
6	如果在项目中使用自己的标准函数(*.fct)，必须将备份的标准函数复制到缺省 WinCC 路径\Siemens\WinCC\aplib 下，然后在函数目录树中表示。 • 随着项目的打开，启动*全局脚本*编辑器。 • 使用选项 → 重新生成头文件 菜单条目重新生成声明结构。 现在，在函数目录树中就能看见新的函数。
7	激活项目并检查是否正确启动。

3.3.9 重新使用 - 将项目部分传送至新项目或现有项目中

重新使用数据的原因

可以通过不同方法生成 WinCC 项目。最重要的方面是从类似项目重新使用已有的项目部分，或传送预组态实例项目的数据。

项目组

由于最后需要再次将 WinCC 项目合并为一个项目，因此可以为组态组指定相似任务。
WinCC 项目由单个文件(例如画面)和数据库中的组态数据组成(报警记录，变量管理器)。

数据库中的数据

存储在组态数据库中的数据不能创建在两个独立项目内随后一起合并。
因此在组态数据库信息时(例如报警记录的结构)，必须创建用于此组态类型的基本项目。在每次修改数据库前(对于中间步骤也如此)，必须备份基本项目。如果进行修改时出现错误，就可以重新获得修改前的状态。

注意：
注意数据库中所作的每次改变都会影响数据库的结构和存取。许多不必要的修改(可能是删除)将会导致 WinCC 数据库不再被最优化组态并因此导致性能上的损失。

单用户系统

当在基本项目上执行下一组态步骤(例如报警记录)时，决不能对 WinCC 单用户系统的其它地方的数据库进行修改(例如归档变量记录)。

多用户系统或多客户机

相反，如果在多用户系统或多客户机上组态项目，则可在数据库的不同区域中同时对组态进行改变。例如，正在组态系统的一个人可以编辑报警记录，而同时另一个“系统工程师”可以编辑归档系统(测量值采集)。

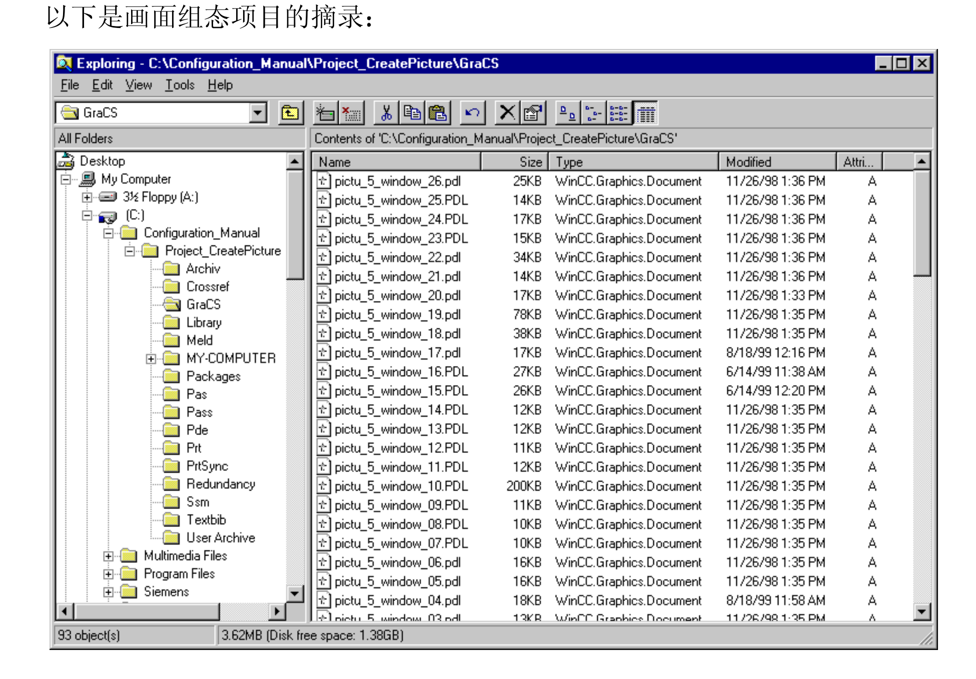
改变项目类型：单用户和多用户系统的转换

任何项目都可以从一个 WinCC 组态系统转化为另一个组态系统。如果预先计划了组态站和客户系统间的组态改变，则不能使用面向多用户系统的 WinCC 指定的元素。例如，如果实际上是为客户端的单用户系统进行组态，则在本地计算机上不能使用内部变量。
在创建新项目时(不管是单用户还是多用户系统)，必须预先知道项目是否基于带有预组态报警记录和/或归档数据的现有的基本项目。可以通过重新组态，或如果可能的话通过导出 - 导入来将存储在数据库中的数据传送到其它项目。

3.3.9.1　画面的传送

任何时候都能够传送已组态的画面。可以使用 Windows 资源管理器(当必须复制多个画面文件时比较有用)将画面文件(*.pdl)直接从源文件夹复制到 WinCC 项目路径\GraCS 下的目标文件夹中。

以下是画面组态项目的摘录：



也可以通过菜单文件 → *在图形编辑器中打开*画面文件(*picture.pdl*)传送画面。 然后，通过菜单*文件* → 另存为将画面*保存*在当前画面文件夹(*\GraCS*)下。如果是以画面文件为基础就适合使用此过程，并将立即进行自定义。

画面中的参考

- 数据类型区域的结构
- 内部或外部变量
- 系统变量
- 作为位图或图元文件存储的画面对象(例如对于状态显示或图形对象)
- 作为图形或过程框或显示窗口使用的其它画面
- 使用的项目函数
- 访问权限

还必须定义下列参考内容：

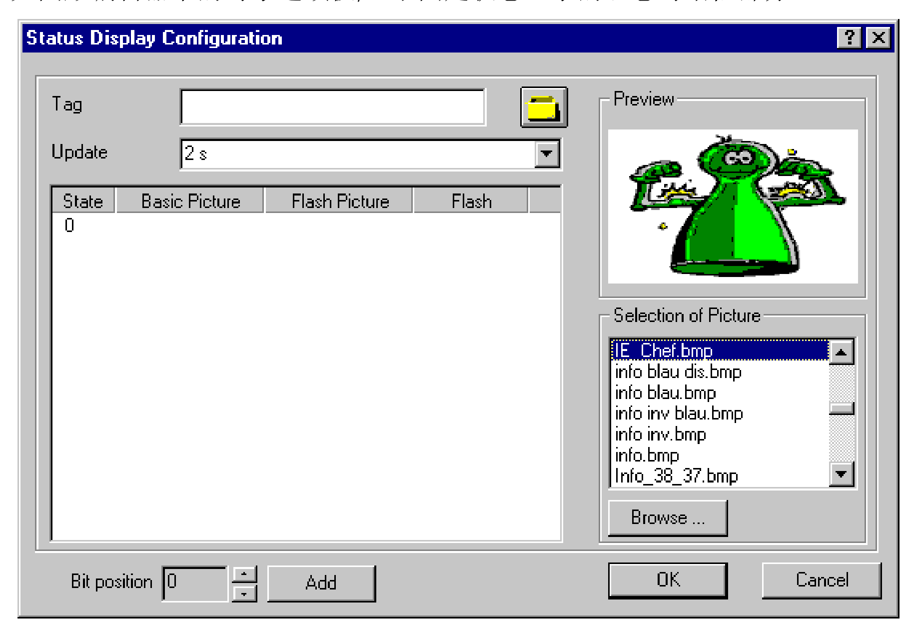
- 数据类型下的结构定义，例如控制器或自定义对象的模板结构
- 通讯通道的定义和与变量定义的逻辑链接(可能与变量组)
- 内部变量或系统变量名称的定义(以@开头)。
- 通过从*GraCS*文件夹复制画面元素(*.gif 或*.emf)来传送它们。
- 从\GraCS 文件夹中通过复制其它画面文件(**.pdl*)来传送画面窗口内容
- 必须将使用的项目函数从源项目复制到新建项目的\library 文件夹。此外，这些函数必须通过再生成头文件菜单存储在全局脚本编辑器的函数树中。这个过程已在 C 脚本的开发环境一章中进行了详细描述。

在用户管理员编辑器中定义将要使用的访问权限。必须为组规定定义访问权限(例如对于控制按钮)。

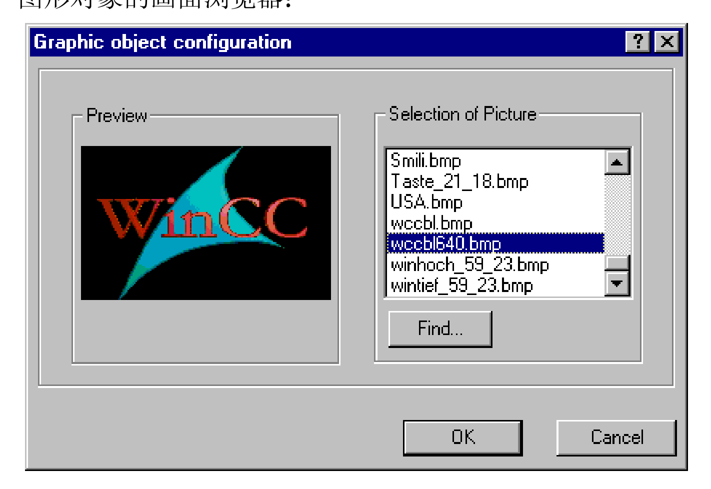
3.3.9.2　符号和位图的传送

复制

状态显示的符号或画面文件中的图形对象可以作为项目画面文件夹中的独立文件存储。 可通过将期望的符号文件(*.emf 或*.gif)复制到新建项目的目标文件夹\GraCS 来完成。这些画面现在立即存在于状态显示或图形对象的选择列表中(参见图形编辑器中的对象选项板)。下图是状态显示的组态对话框部分：



图形对象的画面浏览器：



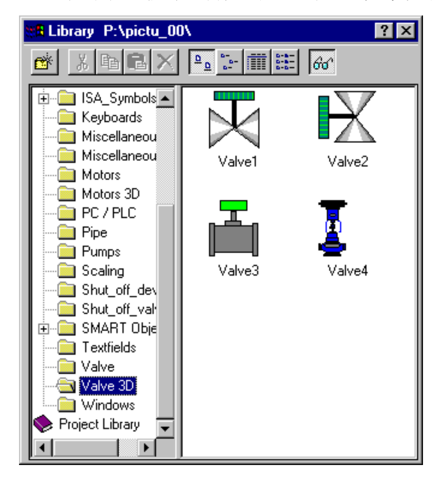
导入

可以使用刚才描述的方法将符号集成到画面中，或通过*插入* → *导入*菜单将符号直接复制到正在编辑的图形画面中。对于后面这种情况，不一定要复制文件，可以通过访问源项目的路径(*GraCS*)直接导入想要的符号，然后选择所期望的符号文件(*.bmp、*.emf、*.wmf)。符号在导入后将立即在画面中(左上方)作为对象显示。
如果符号存储在项目库中，则通过全部传送可以在其它项目中使用该项目库。在下面的章节中描述了如何来执行上述情况。

3.3.9.3 传送项目库(带有预组态符号和自定义对象)

全局库

如果符号存储在项目库中，通过将文件 *library.pxl* 复制到*Library 路径*下就可以将这个库用于另一个项目。
可于任何时侯在新建项目中继续使用预组态的块。



注意：
请考虑已链接的符号可能涉及到一些不可用的参考(或变量)，需要首先对其进行定义。根据这些符号的组态，可能必须调整动作或相关的链接。因此在从库中使用符号后，要检查哪些属性/事件链接已经存在并且它们是否需要调整。

各个符号

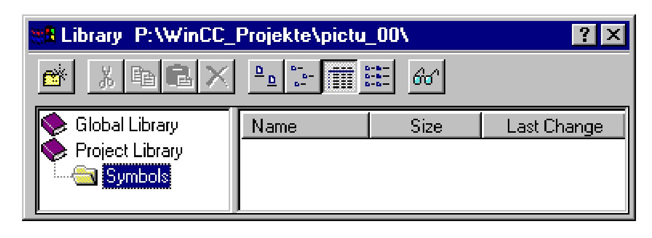
如果仅在新建项目中使用一些项目库中指定的符号，则可以单独导出它们(符号文件*.emf)。

步骤	过程：传送符号
1	打开库。
2	使用 选择期望的符号并按住鼠标键将符号拖到画面中(拖放)。
3	通过*文件* → *导出*...菜单，打开用来保存符号的对话框。 
4	保存符号。

新建项目库

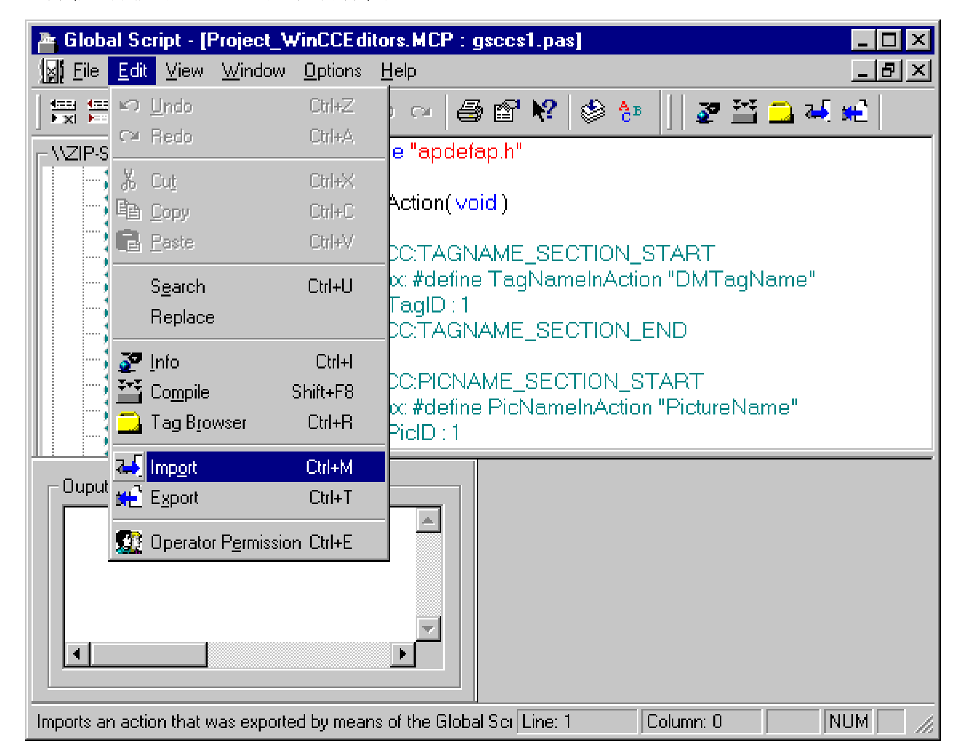
这些导出的符号现在可以作为独立的符号文件使用并可通过导入来单独使用。如果这些符号在项目中频繁使用，可以再一次集成到新建的项目库中。可以通过调用符号库，特别是项目库来进行。

创建自己的符号文件夹:例如借助库窗口工具栏的文件夹图标通过拖放将已导入的符号复制到该文件夹中。可用这种方法来传送项目的部分符号并添加其它指定的符号，以再次创建新的项目指定的库。



3.3.9.4 动作的传送

将需要在项目中频繁使用的动作或将要在不同项目动作间复制的动作存储为独立的文件。这些文件存储在\GraCS 文件夹下，其扩展名为.act。任何时候都可以通过将它们从源文件夹复制到目标文件夹来进行传送。
从 C 动作编辑器中通过工具栏按钮导出动作将动作文件存储到由用户命名的目标文件(具有扩展名.act 表示动作)中。



通过导入动作工具栏按钮，将所存储的动作文件传送到新建项目画面中的对象动作。请参见 C 脚本的开发环境一章的描述。

注意：
经常使用的动作也可以定义为项目或标准函数。

3.3.9.5 变量的传送

可以使用一些不同的方法来补充 WinCC 的变量管理器：

- 读取 S5 数据变量或 S7 数据变量可使用向导(动态向导)
- 传送 S7 变量可通过 PCS7 映射程序
- 使用程序 *Var_Exim* 导入和导出文本列表
- 交互式访问数据库表(变量表)
- 在专门编制的动态向导或程序中使用 WinCC API 函数在变量管理器中生成新的数据

最后提到的两个选项需要非常熟悉 SQL 数据库知识和通过应用程序接口进行编程的知识。只有具备这种知识的人员才能够使用它们。

在将数据传送到目标项目前，需要说明目标项目的地址。如果在 WinCC 的变量管理器中已经有大量变量存在，则应该将 WinCC 变量列表导入到目标项目。内部变量必须始终从 WinCC 的变量管理器传送。可以通过 Var_Exim.exe 工具来完成。

使用动态向导传送 S5/S7 数据变量

借助动态向导，可将使用 STEP5/STEP7 软件产生的数据区定义读入 WinCC 的变量管理器。必须执行下列步骤：

步骤	过程：传送 S5 或 S7 数据
1	备份项目数据。在数据库中做相应改变。
2	使用 STEP 软件导出分配列表。创建名为 *prj_zuli.SEQ* 的文件。
3	删除所有不要求从该导出文件导入 WinCC 的特殊符号(例如对于程序调用)。可以使用典型的文本编辑器，例如写字板来完成。分配列表不能包含任何空白行。
4	打开 *WinCC 资源管理器*中的目标项目。项目必须处于组态模式下(运行系统没有激活)。
5	打开*图形编辑器*。在任意画面中，调转到*动态向导*(通过查看 → *工具栏*...显示)，并选择*导入功能*标签。从此处，选择*导入 S7 - S5 分配列表*的功能。然后，必须指定(使用按钮)*源文件(.seq)*及其路径。 ... 还需指定逻辑连接，在其中将放置对分配列表的变量描述。 现在将数据输入 WinCC 变量管理器中。用于 WinCC 项目的变量名称必须是唯一的。将变量添加到(如果可能)已存在的 WinCC 变量管理器。变量名称对于这个过程非常关键。

使用帮助程序传送变量

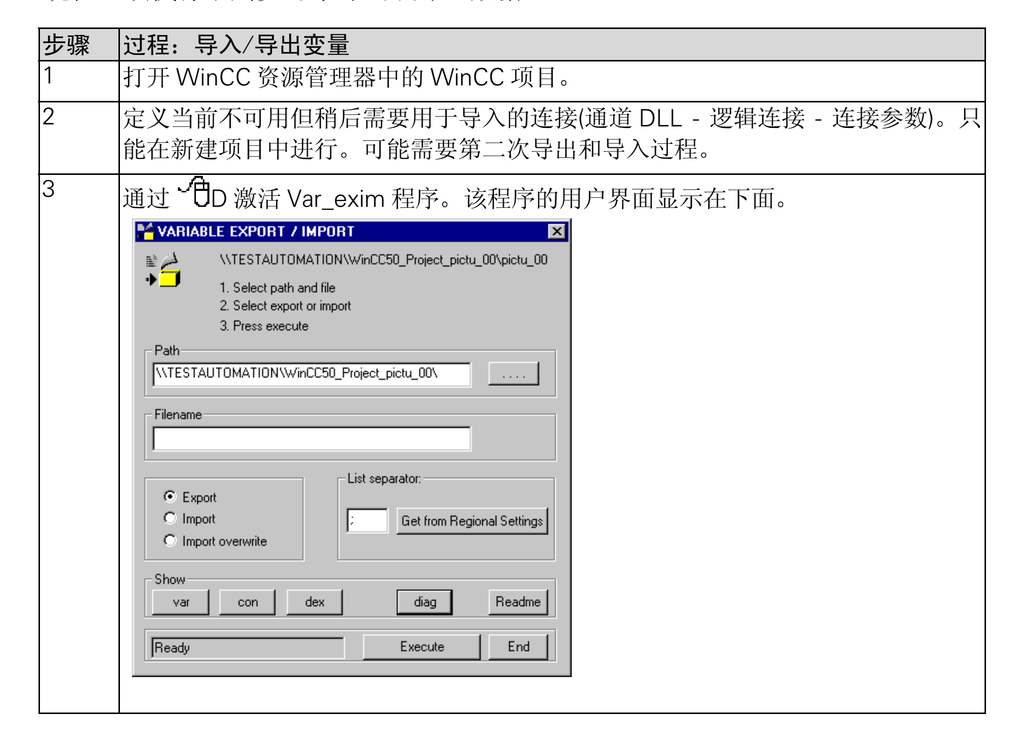
在将连接(通道 DLL、逻辑链接和连接参数)导入目标项目前，必须已进行了定义。

注意：
为了实现 WinCC 数据库中连接和数据条目的自动生成，可以通过 WinCC API 接口编译特殊程序来自动执行这类定义。用这种方法，能够自动添加已存在的设备数据。必须由 WinCC API 编程专家或 SQL 编程专家来编写程序。

可以在任何时候将定义在变量管理器中的变量作为文本文件导出，以补充变量列表。然后，必须将生成的数据**导入**回项目的变量管理器。已创建的文件使用 CSV (逗号分隔的数值)格式，可使用任何编辑程序读取并做进一步的处理。

- 为此，在 WinCC 光盘上的\SmartTools\CC_VariablenImportExport 文件夹中可使用一个独立的应用程序。该 Windows 程序作为帮助应用程序提供，用来进行：
- 变量管理器数据的导出
- 已在外部生成变量数据的导入
- 大量数据组态

现在必须执行下列步骤来导出或导入数据：

步骤	过程：导入/导出变量
1	打开 WinCC 资源管理器中的 WinCC 项目。
2	定义当前不可用但稍后需要用于导入的连接(通道 DLL - 逻辑连接 - 连接参数)。只能在新建项目中进行。可能需要第二次导出和导入过程。
3	通过 D 激活 Var_exim 程序。该程序的用户界面显示在下面。

导入和导出

必须执行下列步骤来进行导出或导入：

位置、动作	导入	导出
路径	选择包含用于变量导入的文件的项目文件夹。该选择过程可以通过选择名为*.mcp* 的文件来完成。 包含了用于导入的数据的文件必须位于与项目文件相同的文件夹中。	指定用于变量导出的项目文件夹。该选择过程可以通过选择名为*.mcp* 的文件来完成。
动作	选择*导入*。 如果将要改写现有的变量数据，则选择*导入改写*。	选择*导出*。
执行	单击执行。 随后显示的对话框显示参数设置并在单击确定后执行该过程。 由于要执行检查，因此导入需要花更多的时间。	单击执行。 随后显示的对话框显示参数设置并在单击确定后执行该过程。
状态显示	结束导出/导入	结束导出/导入
变量文件 *Name_vex.csv*	导入的基础： 由标题和数据记录组成。	生成的变量列表作为文本存储在该文件中。这个文件可以通过单击 var 按钮来打开，或使用文本编辑器(记事本)或 EXCEL 进行编辑。
变量文件 *Name_cex.csv*	导入的基础： 由标题和数据记录构成(结构组件)。	这个文件包含了在变量文件中所涉及的当前已组态的连接。这个文件可以通过单击 *con* 按钮来打开或使用文本编辑器(记事本)或 EXCEL 进行编辑。
数据结构文件 *Name_dex.csv*	导入的基础： 由标题和数据记录构成。	如果包括具有数据结构类型的变量，则还将生成包含结构信息的文件。可以使用文本编辑器(记事本)或 EXCEL 编辑其内容。
诊断文件 *Diag.txt*	诊断文件包含不能被导入的变量的信息。	

通过单击*结束*按钮可以退出该程序。

变量列表

下表描述了变量列表的结构。

域	类型	描述
Tag Name	char	变量名
Conn	char	连接
Group	char	组名
Spec	char	内部变量或地址(与连接类型匹配)
Flag	DWORD	
Common	DWORD	
Ctype	DWORD	变量类型 1 BIT 2 SBYTE 3 BYTE 4 SWORD 5 WORD 6 SDWORD 7 DWORD 8 FLOAT 9 DOUBLE 10 TEXT_8 11 TEXT_16 12 原始数据类型 13 域 14 结构 15 BITFIELD_8 16 BITFIELD_16 17 BITFIELD_32 18 文本参考
CLen	DWORD	变量的长度
CPro	DWORD	内部或外部变量
CFor	DWORD	格式转换
Protocol		
P1	BOOL	上限错误
P2	BOOL	下限错误
P3	BOOL	转换错误
P4	BOOL	写错误
P5	BOOL	
P6	BOOL	
L1	BOOL	上限错误的替换值
L2	BOOL	下限错误的替换值
L3	BOOL	起始值

域	类型	描述
L4	BOOL	连接错误的替换值
L5	BOOL	上限有效
L6	BOOL	下限有效
L7	BOOL	起始值有效
L8	BOOL	替换值有效
LF1	double	上限
LF2	double	下限
LF3	double	起始值
LF4	double	替换值
Scaling		
SCF	DWORD	当定义定标度时数值为 1
SPU	double	取值范围，过程最小值
SPO	double	取值范围，过程最大值
SVU	double	取值范围，变量最小值
SVO	double	取值范围，变量最大值

连接列表

域	类型	描述
ConName	char	逻辑连接名称
Unit	char	通道单元
Common	char	常规
Specific	char	指定的连接参数
Flag	DWORD	

数据结构列表

域	类型	描述
DataStructure	short	数据结构名称或组件名称
Type ID	short	标识(在变量列表的 Ctype 下使用)
Creator ID	short	

为了能继续处理 EXCEL (版本 7.0 或 8.0)中的文本列表，必须打开文件类型为文本文件[*.prn、*.txt、*.csv]的导出文件。

指令

使用以下特殊指令来采用文本列表中的变量数据：

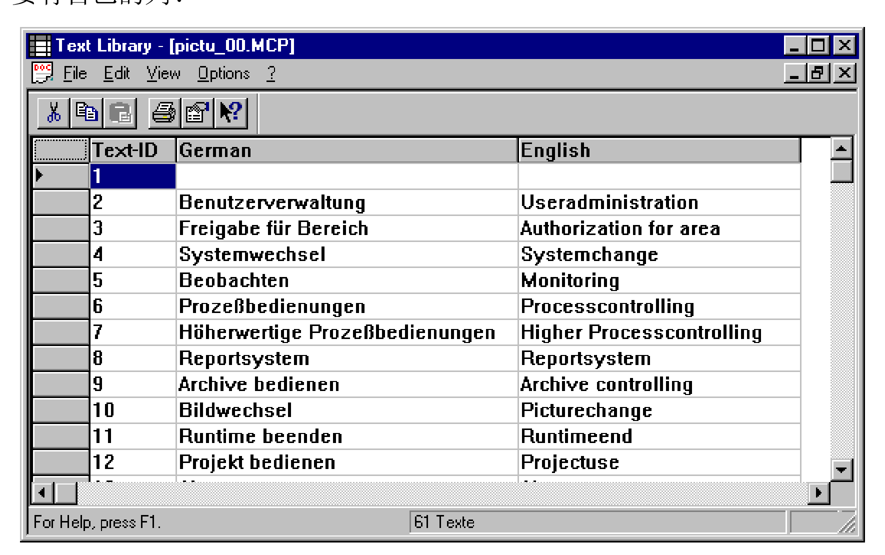
类型	文本列表中	WinCC 中
连接	常规描述	**如果不可用**，则必须**重新定义逻辑连接！**
	通道 DLL 的特定描述	**如果不可用**，必须**重新定义通道 DLL！**
变量组	没有组信息 如果在不包含任何变量的项目中定义了组，则也不导出这些组。	当组的第一个变量生成时，将自动生成组信息。
变量	常规描述	
	特定描述 各自的通道 DLL 或内部变量	通道 DLL 或内部变量
	导出期间，丢失部分以*代替。	不导入丢失了特定描述的变量！
数据结构类型的变量	对应于结构列表的数据结构定义的分配。	被分配到数据类型。
数据结构定义		必须由组态系统的人员来定义。
限制	通过文本列表不能导出或导入。	必须由组态系统的人员来定义。

在**开始**导入已编辑的变量或新变量前，首先要进行项目的**数据备份**，这是因为将会修改数据库。这些数据库的改变不能保留在 WinCC 中。

3.3.9.6 多语种文本(来自画面、消息中)的传送

画面的多语种文本

通过复制画面本身可以将存储在画面中的多语种文本传送至新建项目。
必须将各自的语言添加到新建项目的文本库中。检查文本库中的设置。每种语言要有自己的列！



是否只应有部分语言被传送到新建项目中？
由于存储了每个画面中全部的文本信息，需要在每个画面中执行各自语言的重新组态。应该考虑是否有必要删除已组态的文本。只能通过特殊组态键切换到运行模式从而能够进行项目限制。如果仍然要删除已包含的语言文本，推荐使用帮助工具语言进行导出和导入。

带有文本参考的画面的传送

如果文本参考用于传送的画面，同样必须传送下列数据：

- 来自 WinCC 项目变量管理器的相关变量(导出或重新定义)
- 来自文本库的文本
- 必须提供带有有效文本标识号(文本 ID)的文本参考变量。检查文本 ID 是否仍然与相关的文本一致。

从文本库传送文本

如果仅仅传送来自文本库的部分文本，必须相应地调整文本 ID。可以通过*文本库*编辑器的导出/导入机制传送文本库的文本。

3.3.9.7 消息的传送

消息(报警)的信息基础要求

- 修改和
- 对卷执行大量的组态工作(大量数据)。
- 因此，需要经常用到传送来自先前项目的消息数据的选项。根据消息数据资源可以使用下列方法来传送消息：
- 从原有的系统中传送已组态的消息信息(例如 COROS)
- 从现有的 WinCC 项目导入(单个)消息
- 从概念阶段导入消息信息

传送来自 COROS 的消息

上面提到的资源过程如下所示：

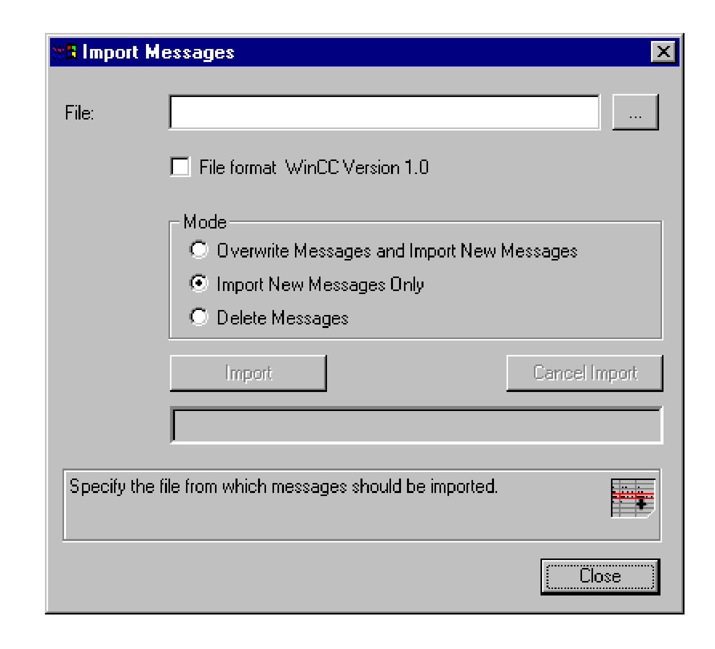
步骤	过程：传送现有的 COROS 消息文本
1	在 COROS 中，导出消息信息(*mldtexte.txt*)
2	在 WinCC 中，使用动态向导导入功能 → 导入消息来导入该消息文件。数据被导入当前的 WinCC 项目中。

传送来自 WinCC 项目的消息

如果从现有的项目中传送消息，首先必须说明是否已经用同样的方法在数据库结构中建立了目标消息系统(报警记录)。例如，在用户和过程值块中可以看见差异。如果可能，立刻根据序列(还可根据文本元素的长度)直接组织目标数据块。否则必须在导入前在各列中进行调整。

步骤	过程：传送现有的 WinCC 消息文本
1	在当前项目中，打开*报警记录*编辑器。
2	通过选择消息 → *导出单个消息*菜单对消息进行导出。
3	指定目标文本文件，将要导出的消息信息存储在此文件中。 在选择处，使用判断标准(例如编号、消息等级)选择将要导出的消息。
4	通过单击*导出*启动导出过程。现在创建一个文本文件，它包含由逗号分隔的消息信息条目。
5	关闭当前项目并打开一个新建的项目。再次打开*报警记录*编辑器并定义必要的消息级别和消息类型。为了对下列导入保留基本结构，需要为每种消息类型定义一条消息。
6	为了导出这些基本消息，选择 → 导出单个消息。重复步骤 3 和 4。

步骤	过程：传送现有的 WinCC 消息文本
7	例如，现在用 EXCEL 打开源项目的消息文件和目标项目的消息文件。列由逗号分隔。 比较消息块的结构，如果必要可通过重新组织或重新命名列来进行调整。 在带文本 ID 的每个块中输入下标 0。也就是说文本被导入时能在文本库自动组织。决不要保留老的 ID 编号！ 已修改的文件必须再次作为文本文件保存。
8	通过*消息* → *导入单个消息*来开始导入过程。
9	现在指定带有导出消息信息的源文本文件。需要确定导入期间是否要重写已存在的消息。通过各自的消息编号分配消息，消息编号在项目中必须是唯一的。
10	然后导入消息并向现有的消息系统(报警记录)增补已组态的消息信息。检查导入的分配。



注意：
如果从 WinCC V 1.10 项目中传送消息数据，必须注意消息文本文件中的列标题！

通过 EXECL 表格从概念阶段导入消息信息

消息信息已存在于 EXCEL 表格中。通过将列合并成 WinCC 项目的消息结构可以传送这些消息。但这必须通过构建一个 WinCC 消息文件来完成。按下列步骤生成这个文件：

步骤	过程：创建消息结构
1	*在 WinCC 资源管理器*中打开新建的 WinCC 项目。打开*报警记录*编辑器并定义必要的消息块、消息级别和消息类型。为了对下列导入保留基本结构，需要为每种消息类型定义一条消息。
2	为了导出这些基本消息，选择*消息 → 导出单个消息*。
3	指定目标文本文件，将要导出的消息信息存储在此文件中。
4	通过单击导出启动导出过程。将创建一个文本文件，它包含由逗号分隔的消息信息。
5	在 EXCEL 中打开目标项目的消息文件和新近创建的消息文件。列由逗号分隔。 在表中为相应的消息级别/消息类型创建复制行。从源数据传送消息文本等，并将其输入到相关的块中。例如：*块 1 → 消息文本*。 将所有的消息行(例如从 1 开始)编号。在 EXCEL 中，消息编号列中的编号方式可以帮助非常快速地进行编号。
6	在带文本 ID 的每个块中输入下标 0。也就是说文本被导入时能在文本库自动组织。决不要保留老的 ID 编号！ 已修改的文件必须再次作为文本文件保存。
7	在报警记录编辑器中，通过选择*消息 → 导入单个消息*来启动导入过程。
8	现在指定带有导出消息信息的源文本文件。通过重写已存在的消息的方法来定义参数。通过各自的消息编号分配消息，消息编号在项目中必须是唯一的。
9	然后导入消息并向现有的消息系统(报警记录)增补已组态的消息信息。检查导入的分配。

3.3.9.8 测量值的传送

由于测量点的规定、过程值归档和用户归档的定义及其属性直接集成到数据库结构中，因此如果不直接访问数据库(这需要具备一定的数据库知识)就不能传送测量值。也就是说，必须重新组态这些归档和测量点，或者在组态开始时通过复制整个基本项目来自动传送数据。

3.3.9.9 打印布局的传送

将期望的打印布局(*.rpl* 用于页面布局或 *.rp1* 用于行布局)从源文件夹复制到新建项目的*PRT* 文件夹。

3.3.9.10 全局动作的传送

将期望的全局动作或后台动作 *.pas* 从源文件夹复制到新建项目的*Pas* 文件夹。

3.3.9.11 项目函数的传送

将期望的项目函数*.fct 从源文件夹复制到新建项目的\Library 文件夹。通过全局脚本编辑器中的选项' 再生成头文件菜单使项目了解这些函数。有关这个主题的详细描述请参见 C 脚本的开发环境一章。

3.3.9.12 标准函数的应用

与项目函数相比，标准函数不必复制。由于工作站上所有的 WinCC 项目都知道这些标准函数，因此可以立刻在项目中使用。

3.3.9.13 用户管理器的传送

由于用户组、用户和访问权限的规定及其属性直接集成到数据库结构中，因此不能传送用户管理器。也就是说，需要重新组态。
对此仅有一种方法：在组态开始时，通过复制整个基本项目来启动自动数据传送。

3.3.10 无鼠标时的操作

在大多数情况下通过鼠标来控制 WinCC 中的设备画面。鼠标点击是最频繁使用的方式，可通过大量的变化形式(左或右鼠标按钮的单击或释放)来获得不同的动态类型。还可以使用鼠标和键盘或仅仅是用键盘来控制系统。例如操作面板只能通过键盘控制。

3.3.10.1 通过键盘的操作

键盘提供下列输入选项：

- 功能键 F1 到 F12
- 特殊功能键(例如操作面板功能键 SF10)
- 标准键盘输入
- 使用光标键或特殊键移动到 I/O 域或控制按钮处

对不用鼠标的控制动作的组态必须分别考虑下列组态区域：

- 设备画面中的控制按钮(例如改变画面)
- 通过功能键
- 通过特殊键
- 通过标准键
- 任意键控制
- 通过控制对象在各处移动
- 设备画面中的输入域
- 输入/输出域
- 特殊的输入对象(复选框...)
- 报警记录(消息窗口)
- 通过功能键控制动作
- 通过特殊组态键控制动作
- 变量记录(趋势或表格窗口)
- 通过功能键控制动作
- 通过特殊组态键控制动作
- 通过键启动打印作业
- 通过键盘登录或退出

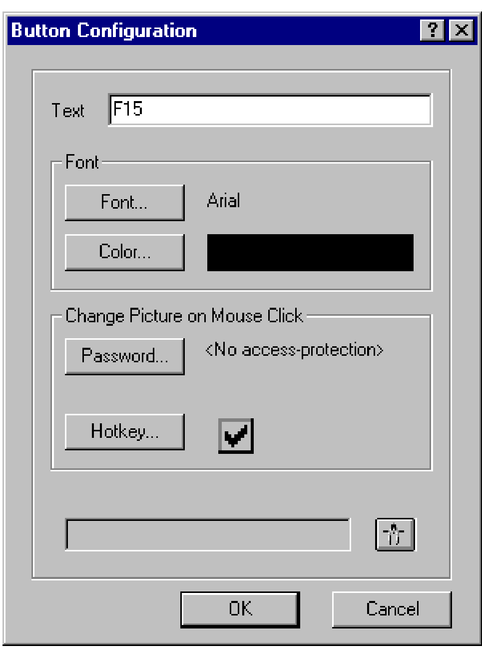
在典型的 *Windows 样式*中组态控制按钮。因此就能在对象选项板中找到标准 Windows 控制按钮。在任何时候都可以将其它图形元素添加给此按钮。

控制按钮



通过功能键控制动作

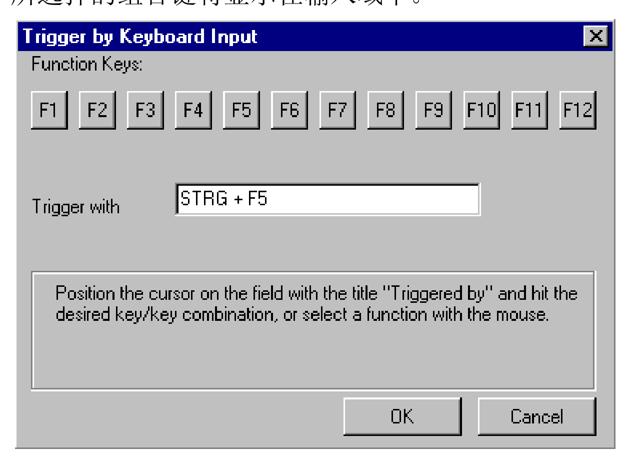
标准键盘上的功能键 F1 至 F12 经常用作设备画面体系中用于改变画面的控制按钮的(附加的)键盘动作。可以在任何时候将这些功能键作为热键分配给已组态的 Windows 按钮。热键提供了最快捷的方式来启动为其分配的功能。



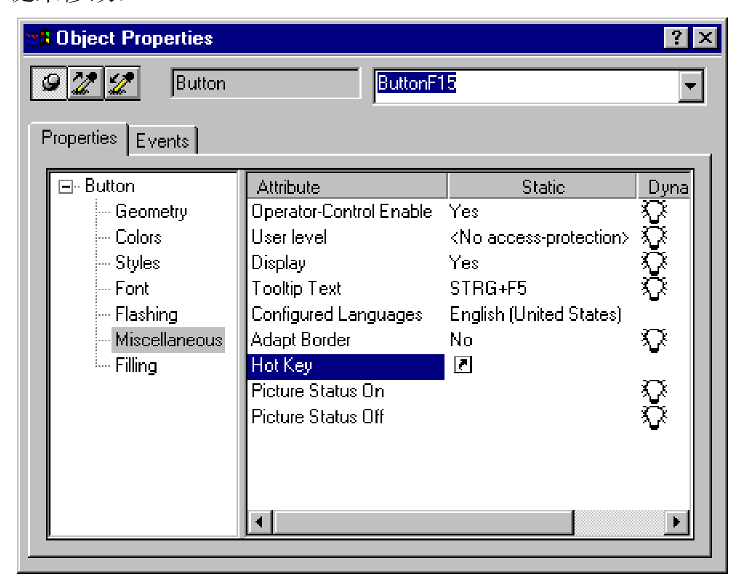
例如可将热键分配给上述功能键。这些键作为选择按钮已经显示在组态对话框中。

如果需要将一个键与 SHIFT 键或 CTRL 键组合，则只需简单地通过按下相应的键来在输入域中直接输入期望的键序列(例如 SHIFT+F2)即可。不需要输入任何特殊的代码。

所选择的组合键将显示在输入域中。

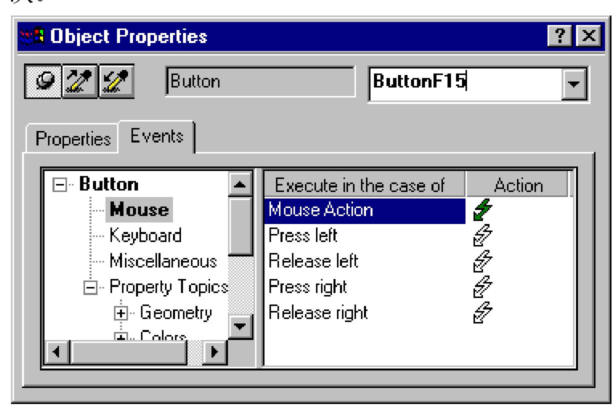


在对象属性中存储键的选择并可通过组态对话框或直接通过属性中的*其它* → *热键*来修改。



热键的动作

通过输入功能键(热键定义)触发的动作必须存储在 Windows 按钮事件下的鼠标动作中。当释放鼠标按钮时触发事件，但仅当鼠标指针定位在对象上按下并释放时才会触发。如果没有动作存储在*鼠标动作*事件下，例如仅存储在(相似)事件按*左键*下，动作就**不能**由功能键触发！注意在组态时，功能键只能在画面中使用一次。



特殊的功能键

如果使用特殊的功能键来控制画面，例如操作面板按钮 F13、S1 等，需要将这些键重新分配给组合键。例如键 F13 能重新分配到组合 SHIFT+F1。除了使用上述可视化结构中所选择的组合键，还能为特殊设备专门定义各种组合。可以找到依赖于各自使用的设备特殊的键盘设置。例如，提供文件 FI25.key 用于工业 PC 键代码的设置。这些设备指定的文件用于存储功能键的代码。在调整设备键盘定义(每个功能键有其对应的十六进制代码)并激活新的键盘代码后，设备画面中的这些键能用于画面中的工作。

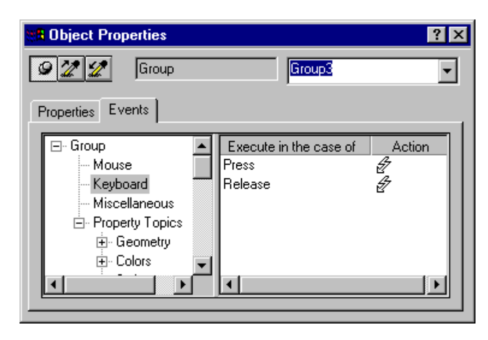
标准键

如果没有将动作分配给功能键输入而是分配给标准键盘上的键，例如字母 m，该键就作为热键存储在 Windows 按钮对象下。

任意键控制

可以自己创建设备画面的控制按钮。例如可以在 3D 按钮或用户对象下的用户库中查找其它控制按钮。
不基于 Windows 按钮的自行设计的对象不能作为热键组态。通过对象的按钮*事件*，所有其它对象必须被组态或按钮控制。下列键盘事件可用于每个对象：

- 按下
- 释放



按钮事件必须可用于激活键盘控制动作的组态。当从用户库使用预定义按钮时，首先要检查该按钮是否只适合采用键盘工作而不用鼠标。例如对于键盘控制动作，用户对象的移位按钮就不起作用。因此在这种情况下，必须对按钮作出调整。

预组态的按钮控制(例如通过画面体系滚动)可以参见可选包(例如基本过程控制 - 画面目录管理器等)。
如果这些对象之一用作控制元素，则组态触发动作的按钮来响应键盘 - 按下或键盘 - 释放事件。也可以组态直接连接或 C 动作作为动作。
触发按钮事件可以是

- 任意键动作或
- 标准键盘上所选择的键。

如果事件是任意键事件，可以使用*直接连接*。反之如果必须检查指定键输入，则必须使用 *C 动作*。在继续实际的动作序列前，*C 动作*将逐个字符地检查输入的键代码：

3.3.10.2 在控制对象(输入域和控制域)上移动

可以使用鼠标在每个可控制对象上直接单击。对象是否可被控制将通过鼠标指针的改变来显示。如何不使用鼠标来操作这些对象？

Alpha 光标

通过“移动键操作”可以在运行模式下的可控制对象之间移动。区别在：

- alpha 光标对象(I/O 对象)和
- tab 顺序对象

通过 alpha 光标(Tab 键或 SHIFT+Tab 组合键)选择输入/输出对象。
所有控制元素(通过鼠标、键盘或者同时这两者可控制的元素)都能通过 tab 顺序集成到控件中。通过 alpha 光标和 tab 顺序可以把 I/O 域集成到控件中。

TAB 顺序

TAB 顺序(可以通过编辑 → TAB 顺序 → Alpha 光标或 → Tab 顺序来设置)可让用户在运行模式下改变可控制对象跳转的顺序。在运行系统中可以将当前所选择的对象可视化。这就是可关闭的运行光标(计算机属性 → 图形运行系统)。以 Windows 样式组态的按钮被选中时，在按钮内总是显示由虚线构成的矩形。
根据图形运行系统的设置在可控制元素上移动(计算机属性 → 图形运行系统)。

移动	标准键	键设置
向上、向下 向左、向右	光标键或 Tab (向后)或 SHIFT+Tab (向前)	其它键设置可通过*计算机属性 → 图形运行系统 → 光标控制键*完成
Alpha 光标/Tab 顺序	Tab 顺序	使用热键(*计算机属性 → 图形运行系统 → 热键*)或自定义键(使用内部函数 *SetCursorMode*)在 alpha 光标和 tab 顺序之间切换。
在表格中移动(光标组)	常规，即逐行编辑 当光标达到光标组的末尾时，它仍然在此位置。	这个动作可以通过*计算机属性 → 图形运行系统 → 光标控制特性来改变*

输入/输出域

已组态的输入/输出域在选中后可通过键盘直接写入，也就是说可以立即输入新的数据。不能操作纯输出域(属性 → 输出/输入 → 域类型 → 输出)。
根据所组态的属性(*属性 → 输出/输入 → 退出时应用*)，用 ENTER 键确认输入。
反之，如果不保存所作的改变(如果存在)时用取消键(ESC)退出输入状态。

其它输入对象

Windows 应用程序除了提供典型的模拟输入域外还提供其它输入选项。在 Windows 对象的对象选项板下可以找到这些特殊的对象

- 复选框
- 单选按钮

通过空格键可以设置复选框或选项钮的各个选择域，通过向上/向下键(例如方向键)可以在框内的各个组件中移动。这就是缺省设置的键赋值。
另一个输入对象是文本列表对象。可以通过根据已组态的条目而打开的列表来选择：
同样可以通过标准键盘控制该对象。不需要为键盘控制动作存储特殊的组态。
按 ENTER 键打开列表，通过向上/向下键在列表中移动并按 ENTER 键以确认当前选择。
通过 WinCC 中的 OCX 元素可以使用更多的输入对象。这些对象的组态和控制取决于已专门为对象定义的事件和属性。必须清楚理解这里的每种情况。

3.3.10.3 工具栏按钮的报警记录功能键

在消息窗口的工具栏中可以设置不同的控制按钮，可通过鼠标(标准)对其进行控制。
消息窗口中最常用的控制动作是

- 选择确认消息
- 消息列表中上/下移动
- 在消息列表中滚动



当打开消息窗口或执行进一步的控制动作时，控件必须位于消息窗口中而不能在主窗口中。根据当前的可控性，按钮控件(或功能键)对主窗口的功能键栏或消息窗口存储的按钮控件起作用。例如通过设置窗口区域中当前的控制焦点可以达到该目的。通常使用鼠标单击来设置控制焦点。
使用键盘可以通过下列组态途径在消息窗口中设置焦点：

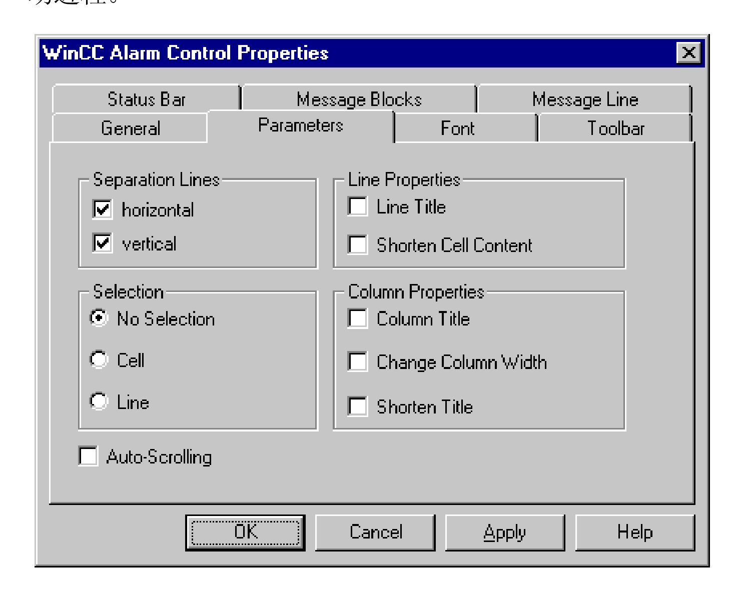
- 通过热键切换窗口
- 通过控制按钮设置焦点或
- 在打开画面时对消息窗口中已定义的元素直接设置焦点。

在图形运行系统的启动参数中对通过快速控制动作(热键)转换(切换)到消息窗口进行了定义，还可采用同样方法进行所有窗口的切换(即在趋势窗口中)。在计算机属性 → 图形运行系统 → 热键 → 置前窗口下输入组合键(例如 CTRL+W)。

一旦打开消息窗口，通过这种键顺序可以直接激活工具栏中的按钮。
然而，通过内部函数 Set_Focus 可以在消息窗口中直接设置控制的焦点。因而用于按钮控件或打开画面命令的 *C动作(画面对象 → 事件 → 其它 → 打开画面)*就会影响消息窗口控件的激活。
关于画面焦点的函数可以从*内部函数 → 图形 → 设置 → 焦点*处获得。例如，设置控制焦点时，可以调用下面这个函数：

```
Set_Focus(lpszPictureName,lpszObjectName);
```

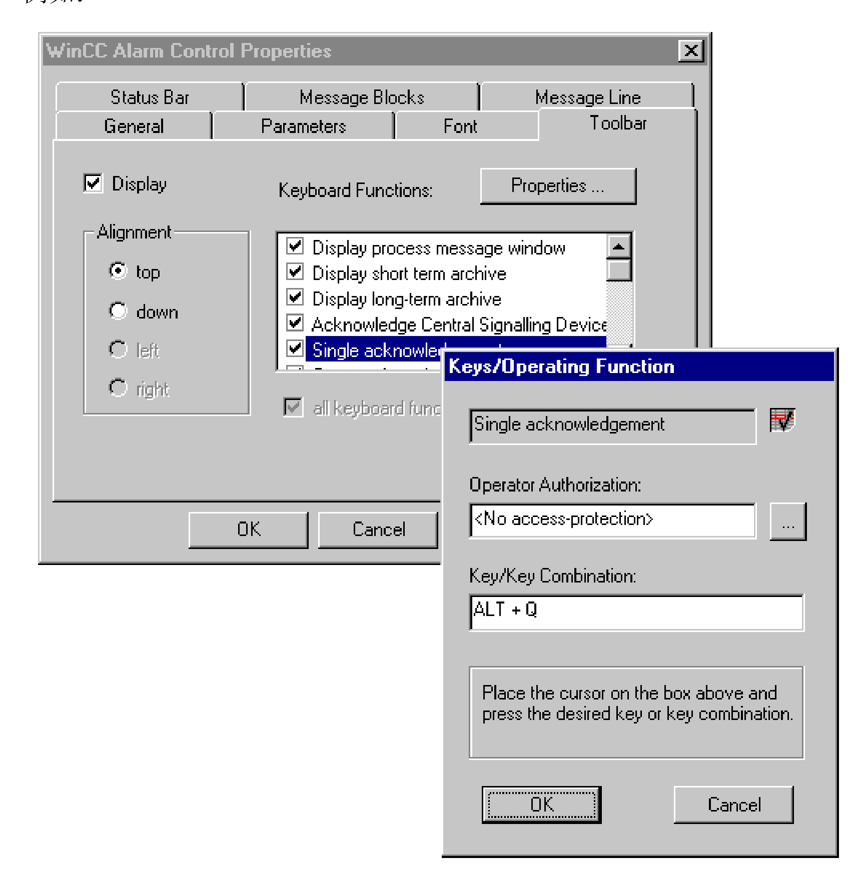
作为参数，必须指定主窗口(画面名称)和报警控制(对象名)的名称。
通过选择消息行选择消息窗口中的消息。当打开消息窗口时，光标始终定位于最新的消息(消息画面中最近的消息)。选择消息窗口中的消息或进行滚动都将取决于滚动进程的激活。
可以通过工具栏中的按钮或者直接在 WinCC 报警控制的组态中打开/关闭画面滚动进程。



如果激活消息窗口中的滚动和控制焦点，可以进行如下移动：

移动	标准键	键设置
在消息列表中向上、向下移动	方向键	单个消息行
消息列表中的开始、结束	Pos1 键、结束键	开始或结束消息列表
滚动	滚动键(页上移/页下移)	多个消息行

除了能激活工具栏上的按钮(例如所选消息的确认)外，还能通过功能键定义控制动作。
对于工具栏上显示的每个按钮，可以在属性中组态相应的键盘控制。
例如：



这些组态步骤可以对消息窗口使用的每个按钮分配组合键。

3.3.10.4 专为设备设计的报警记录 - 工具栏按钮

所有的工具栏按钮都由 WinCC 指定且不能改变。如果要为组态的设备指定某个按钮布局，则必须使 WinCC 工具栏无效(即不使用工具栏)并自行设计相关的按钮。可以设计所有这些新建按钮对象来满足客户的愿望，例如给定的图标。
分配给某个特定按钮的功能必须仍然作为相关的动作来组态。在相关事件的 C 动作(例如按下按钮)中，必须从函数目录中选择相应的标准函数。
为按钮控件提供的函数位于标准函数 → 报警中。该列表包含与工具栏上每个按钮相对应的函数。例如，通过下列函数调用单个确认按钮：

```
AXC_OnBtnSinglAckn(lpszPictureName,lpszObjectName);
```

作为参数，必须输入报警记录控制的窗口标题。
这些动作也可用于通过鼠标控制自行设计的按钮。
报警系统的选项包中提供了一些按钮实例(例如基本过程控件 - 蜂鸣器确认等)。

3.3.10.5 工具栏按钮的变量记录功能键

在用于显示测量值的变量记录趋势或表格控件(趋势和表格窗口)中，为工具栏组态各种可使用鼠标来操作的控件按钮。
趋势窗口中最常用的控制动作是

- 滚动测量值(时间坐标轴)
- 选择时间范围
- 选择趋势
- 使用读标尺

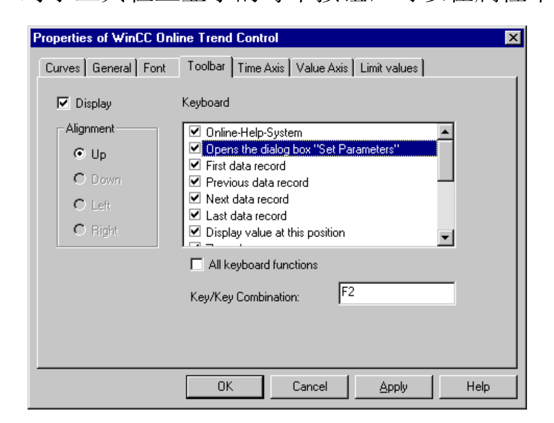
在打开趋势窗口后，根据组态显示当前的趋势曲线。

打开趋势窗口时，控件必须位于趋势窗口中而不是主窗口中。根据当前的可控性，按钮控件(或功能键)对主窗口的功能键栏或者趋势或表格窗口存储的按钮控件起作用。例如通过设置窗口区域当前的控制焦点可以达到该目的。通常用鼠标单击来设置控制焦点。
使用键盘可以通过下列组态途径在趋势或表格窗口中设置焦点：

- 通过热键切换窗口
- 通过控制按钮设置焦点或
- 当打开画面时在趋势窗口中将焦点直接设置到已定义的元素

可以在描述报警记录的章节中找到这些不同变量的执行。
除了通过标准鼠标动作激活工具栏上的按钮以外(例如时帧的选择)，也可以通过功能键定义控制动作。缺省状态下由功能键 F1 至 F10 占用各个按钮。

对于工具栏上显示的每个按钮，可以在属性中组态相应的键盘控制。例如：



这些组态步骤可以对趋势或表格窗口所使用的每个按钮分配组合键。因此还必须定义趋势或表格窗口的按钮控件。
一旦激活各个功能键，就可以在趋势窗口中使用下列标准键操作：

移动	标准键	键设置
读标尺	方向键	左移或右移读行
放大	选择要放大的部分	设置和激活代替鼠标的辅助输入(参见系统设置)。 INS 和方向键可以定义缩放的窗口。
对话框，例如归档变量选择	Tab 键	在输入域间移动
	方向键	在变量选择或标签选择内移动
	+键(-键)	显示或关闭归档变量的目录
	空格键	选择或撤消选择
	ENTER 键	确认并退出对话框
	ESC 键	取消对话框

为设备专门设计的变量记录 - 工具栏按钮

所有的工具栏按钮都由 WinCC 指定并且不能改变这些按钮的设计。如果要为组态的设备指定某个按钮布局，则必须使 WinCC 工具栏无效(即不使用工具栏)并自行设计相关的按钮。可以设计所有这些新建按钮对象来满足客户的愿望，例如给定的图标。
分配给某个特定按钮的功能必须仍然作为相关的动作来组态。在相关事件的 C 动作(例如按下按钮)中，必须从函数目录中选择相应的标准函数。
为按钮控件所提供的函数位于*标准函数* → *taglog* → *toolbarbuttons* 中。该列表包含与工具栏上每个按钮相对应的函数。例如，通过下列函数调用读标尺：

```
TlgTrendWindowPressLinealButton(lpszWindowName);
```

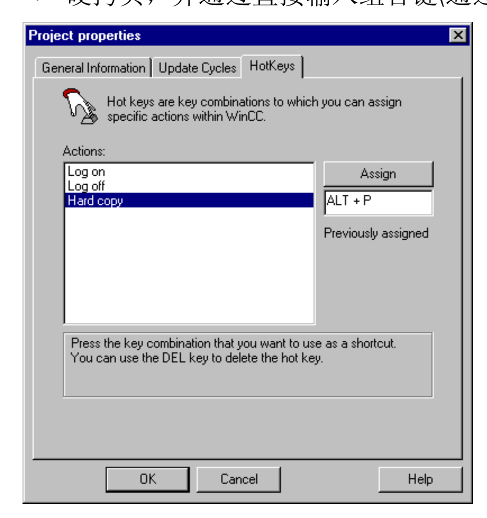
作为参数，必须输入报警记录控制的窗口标题。
这些动作也可用于通过鼠标控制自行设计的按钮。

3.3.10.6 启动打印作业

可以通过一些不同的方法启动打印作业。例如，在 *WinCC 资源管理器*中，通过从选择列表中选择打印作业来直接激活打印作业。也可以在设备窗口创建打印按钮，使用该按钮可以启动打印作业。

该按钮位于消息列表的工具栏中，并且可如前面所描述的将由功能键或自行设计的键来激活。

通过热键可以在每个画面中激活画面内容的打印输出，即“硬拷贝”。在项目属性中对热键进行全局设置。为此，从 WinCC 资源管理器选择项目属性 → 热键 → 硬拷贝，并通过直接输入组合键(通过按下键盘上的相关键)定义热键。



如果在设备画面中为打印已定义的打印作业组态自己的按钮，需要由 *C 动作*来触发动作。正如本章开头所描述的，可以使用例如功能键(参见热键)或键盘动作(例如 D 键)来激活按钮。必须相应地为 C 动作组态上述事件(例如*鼠标动作或键盘 - 按下*)。WinCC 通过*标准函数* → *报表* → *ReportJob* 函数来提供该功能。

```
ReportJob(pszJobName,pszMethodName);
```

这个函数将打印作业的名称作为第一个参数。如果立即启动打印作业，则将“PRINT”作为第二个参数传送，如果将显示打印预览，则将“PREVIEW”作为第二个参数传送。

3.3.10.7 登录或退出

除了为登录或退出过程而组态的热键之外，还可以组态一个键来显示登录的对话框。也可以通过键操作退出。例如，为此必须组态一个通过鼠标动作和键盘两者激活的独立按钮。还可通过按钮的热键属性设置功能键控制动作。本章的开头已详细描述了按钮控件的各种变量。用于登录或退出的函数是 WinCC 应用程序函数。该函数必须作为 C 动作组态。例如，在鼠标动作或按下按钮事件中存储 C 动作。

以下函数用于退出：

```
#pragma code("USEADMIN.DLL")
#include "PWRT_API.H"
#pragma code()

PWRTLogout();
```

由函数 *PWRTLogin()*来执行登录。

下面是如何使用该函数的实例：

```
#pragma code("USEADMIN.DLL")
#include "PWRT_API.H"
#pragma code()

PWRTLogin('1');
```

通过标准键可以控制弹出的对话框：

移动	标准键	键设置
各个输入域	Tab 键(向前)或 SHIFT+Tab(向后) 方向键	左移或右移读行
确认(OK)	ENTER 键	退出对话框并确认输入
取消(中止)	ESC 键	取消对话框或条目

3.3.11 画面模块技术

画面模块技术对于允许已组态画面组件的快速而简单的组态及其重复使用性和可维护性至关重要。

例如，已组态的过程框可用于若干个同类的过程组件(例如阀或控制器)。

将在项目中一起工作并且可视化的控制模块现在可以重新使用已组态的原始画面窗口。按照下列原则进行操作：

- 复制画面窗口并且重新连接变量域
- 使用在调用时分配其变量域的画面窗口(间接链接)
- 应用具有原型的自定义对象和结果对象
- 创建原始画面并且进行集成
- 创建 OCX 画面模块并将它们集成起来作为 WinCC OCX 对象

不同技术的比较

这些技术在以下方面存在很大的区别：如何应用这些技术、组态的复杂性以及这些技术的可能性。因此下面将先对它们进行比较。

类型	优点	缺点
画面窗口的复制	过程简单	必须更改所有的对象链接 对画面组合的更改会引起复杂的后处理
具有间接连接的画面窗口	• 只需用简单的 C 动作对画面窗口组态一次 • 不用复制基本画面窗口就可以重复使用	对画面组合的更改会引起复杂的后处理
自定义的对象	只需使用现有的动态向导对具有连接的对象组态一次	• 对画面组合的更改会引起后处理，即画面再生 • 不能集中更改
原始画面	• 对象只要组态一次 • 可以集中更改	必须具备(良好的) C 语言知识
OCX	• 简单地集成到 WinCC 的组态中作为画面中的对象 • 除了更改对象属性之外，稍后修改 OCX 对象不会引起所生成对象的后处理。 • 高性能 • 其它绘图的可能 • 购入新对象(例如 PCS7 模块)	必须通过编写程序(C++、VB 5)来创建；不能通过 WinCC 组态来创建。

如果项目中仅使用少量的简单画面模块，则前几种变化之一就可足够使项目工程师获得满意。不需要更多的培训就可以实现这些变化。
用户对象特别适合于中低复杂性的简单对象和变量链接。如果能够预见对象需要进行一些改动，则原始画面的概念就会非常有用。
如果图形块很复杂或者需要更全面的处理性能，则最好使用 OCX 技术。将来在此领域中，可用的 OCX 对象会变得越来越强大。
在以后的章节中将描述各种类型的画面模块组态以及如何在设备画面中使用它们。这样将允许用户在项目中形成自己的不同变量及其应用程序的画面。

3.3.11.1 作为画面模块的过程框

为了显示对象(控制器、阀、电机等)的当前状态或者分配设定点数值，在设备画面中显示指定的信息框。这些过程框通常同时包含当前状态(实际值)和设定值，其中设定值可以由特许操作员输入。

创建信息框

此信息框作为画面窗口创建，该画面窗口的组件与相应的(过程)变量连接。

步骤	类型	组态
1	数据结构	通过变量管理器定义画面模块中要使用的数据结构，例如带实际值、设定值和打开-关闭开关的电机。
2	画面模块	使用*图形编辑器*组态显示设备状态的画面(例如棒图和 I/O 域)和控制按钮。画面窗口的尺寸(画面对象属性 - X 变量和 Y 变量)必须与画面窗口的目标尺寸相一致。
3	定义变量	在变量管理器中定义(过程)变量，例如(结构)数据类型为电机类型(用于过程框)的 Motor_T01。
4	变量连接	现在通过将各个画面组件(例如 I/O 域、棒图等)与相应的(过程)变量相连来使各个画面组件动态化。
5	画面窗口	在设备画面中创建画面窗口对象，并通过画面窗口名称属性将它与步骤 2 至步骤 4 下创建的画面窗口内容相连。
6	属性 - 设置	此画面窗口对象不应该在初次打开画面时显示。因此，显示属性必须固定设置为否。 画面窗口(带有 Windows 按钮和标题等)的外观也必须在画面窗口的属性中定义。
7	调用画面窗口	必须使画面窗口可以通过单击按钮或运行设备本身来弹出。设计一个与画面窗口对象的弹出相连的按钮(例如通过直接连接)。

此画面窗口对象、画面窗口内容和相应的画面窗口的调用(按钮)在其它设备的类似窗体中可以再次使用。所有要做的工作只是复制画面窗口对象、画面模块和按钮。参考必须每次都进行调整。画面窗口对象和按钮二者都可以通过拖放到图形库(例如项目库)中来复制。

自定义画面模块

因此使用创建的画面模块时必须执行下列各步骤：

步骤	类型	组态
1	过程变量	为已定义的数据结构定义新过程变量，例如 Motor_T02。
2	画面模块的复制	复制画面窗口内容(Motort02.PDL)，并且更改所有永久存储的参考(例如现在用 Motor_T02.ActValue 代替 Motor_T01.ActValue)。
3	画面窗口的复制	在目标设备画面中复制画面窗口对象(通过从图形库拖放)。在*属性* → *画面名称*(Motor02.PDL)处调整画面窗口内容的参考。
4	按钮的复制	在目标设备画面中复制按钮(通过从图形库中拖放)。在直接连接中调整新画面窗口对象的参考(对象 → 画面窗口 2 → 显示)。

用这种方法可以为每个设备创建各个画面窗口及其内容，而且这些画面窗口及其内容通过复制可以再次使用。正如所看到的那样，必须做的工作是为画面窗口内容调整永久存储的参考。因此，要再次使用可以有一个更为简单的方法，即通过间接寻址。调整所需的工作量应该保持最小。
对此还有另一种解决方法，就是组态与画面窗口没有连接的画面模块。也就是说将画面模块本身组态为设备画面中的非显示对象。然而这也存在很大的缺点，即当更改画面模块时，使用此画面模块的所有画面都必须进行相应的更改。

3.3.11.2 带间接寻址的画面模块

至此画面模块的各个组件已与相应的(过程)变量永久连接。如果不是通过永久组态进行连接而是在运行期间动态地进行连接，则使用创建的画面模块时具有更大的灵活性。(过程)变量的此动态连接通过画面模块中各组件的间接寻址来实现。也就是说(过程)变量没有直接连接；只是与容器进行连接，该容器在运行期间将携带相应(过程)变量的当前名称。

通过此方法可以大大简化画面模块的调整和重复使用特性。

用与上述步骤类似的方法来执行组态。此处是要执行的实际步骤：

步骤	类型	组态
1	数据的规定	一方面用变量管理器定义画面模块中要使用的数据(例如 Motor001_ActValue,Motor001_SetValue,Motor001_Switch)，另一方面规定画面模块中要使用的各组件的名称容器(例如 ActV_Name、SetV_Name 等)。用一个名称来初始化这些变量，例如 Motor001_SetValue。
2	画面模块	用*图形编辑器*组态显示设备状态的画面(例如棒图和 I/O 域)和控制按钮。画面窗口的尺寸(画面对象属性 - X 变量和 Y 变量)必须与画面窗口的目标尺寸相一致。
3	变量连接	现在通过将各画面组件(例如 I/O 域、棒图等)与包含相应变量名称的相应容器变量相连来使画面组件动态化。但是必须在连接中声明，该变量只是实际(过程)变量的名称。通过选中间接寻址列来进行此操作。
4	画面窗口	在设备画面中创建画面窗口对象，并通过画面窗口名称属性将它与步骤 2 和步骤 3 下创建的画面窗口内容相连。
5	属性 - 设置	此画面窗口对象不应该在初次打开画面时显示。因此，显示属性必须固定设置为否。 画面窗口(带有 Windows 按钮和标题等)的外观也必须在画面窗口的属性中定义。
6	调用画面窗口	必须使画面窗口可以通过单击按钮或运行设备本身来弹出。设计一个与画面窗口对象的弹出相连的按钮(例如通过直接连接)。
7	图形库	将画面窗口对象和按钮复制到库中(通过拖放)，以便再次使用。

3.3.11.3 自定义的对象

自定义的对象和相应的动态向导可用于创建易于再次使用的画面模块。复制的画面模块通过使用向导进行简单组态就可以与相应的当前(过程)变量相连。
自定义的对象是由项目工程师设计的图形对象(例如数个对象的组合)，它的大量属性和事件通过组态对话框可以简化为基本的属性和事件。通过相应的向导将此自定义的对象定义为一个原型来使其动态化。
必须采用下列步骤

步骤	类型	组态
1	数据结构	通过变量管理器定义画面模块中要使用的数据结构。
2	画面模块	使用*图形编辑器*来组态具有用户定义属性的自定义对象。

自定义的对象由一组 WinCC 对象组成。最初这些对象并没有组态为动态。选择将要组成自定义对象的所有对象，并调用自定义对象的组态对话框：
在此对话框中声明所有的对象属性都是自定义对象的属性，这些属性稍后将会动态化。对象的基本属性(例如位置和尺寸)已经被存储用于自定义的对象。
通过拖放，可以在对话框中选择该组对象的各个属性并将其作为*用户定义的属性或事件*添加到新的自定义对象中。
每个属性可以由用户分配新的(不依赖于语言)属性名称以及依赖于语言的属性名称(例如用英语组态)。对于不应在属性对话框中可视显示但是脚本中实例要使用的属性，可以通过使用字符@来隐藏。也就是说只可以显示少数(要动态化的)属性和事件。其余将全部隐藏。
现在必须使设计的自定义对象动态化。为此，提供了一个向导：

步骤	类型	组态
3	动态化	调用动态向导为原型*添加动态特性*。 作为模板(也就是原型)，将对象的每个独立属性与相应的已定义数据结构的结构组件链接。 用变量浏览器选择连接的*结构成员*。 然而向导只保存已链接属性(例如数值)的结构组件名称。每个独立的属性必须单独链接。 此对象现在是一个动态对象，但是它只是作为原型进行链接，在运行时不能激活。也就是说在运行期间不能更新。
4	复制到图形库	将此原型作为对象复制到图形库。

例如，将原始的自定义对象复制到图形库中，以便可以反复使用。动态对象的实例之一是 WinCC 库(用户库、自定义的对象、指针工具)中的指针工具。
将原始对象插入目标设备画面中，作为原型的复制品。此复制品现在必须与来自变量管理器的实际(过程)变量链接。

步骤	类型	组态
5	变量	为步骤 1 中定义的数据结构定义(过程)变量；此变量要用于自定义的对象。
6	创建事例	通过拖放将原始对象从图形库复制到设备画面中。 通过动态向导链接原型与结构来连接此对象与(过程)变量： 通过用实际变量链接替换每个属性的原始变量链接，向导自动将所有必需的变量结构组件与原始画面模块的正确属性相链接。 现在就有了一个在运行期间用当前变量值更新的对象。

3.3.11.4 动态事例

除了动态向导链接原型与结构之外，还有一个名为使原型动态化的向导。它与链接到结构有何不同？必须修改哪些步骤？
不同于将对象永久链接到变量，画面模块也可以进行动态链接。也就是说运行的事例首先根据变量的当前内容来设置。例如上述画面模块没有与变量永久链接；变量名称保持动态。那么变量的当前名称必须通过文本变量来确定。包含变量当前名称的此文本变量必须与画面模块相链接。
与永久性的事例(即与结构链接)不同，在组态时必须修改下列步骤：

步骤	类型	组态
1	画面模块	根据如上所述，用图形编辑器组态具有用户定义属性的自定义对象。自定义的对象必须包含静态文本组件，其文本属性作为自定义的组件进行传送。 将属性名称 TagName 分配给此文本属性。此*变量名称*用于(过程)变量的动态链接。
2	变量	为步骤 1 中定义的数据结构定义(过程)变量；此变量要用于自定义的对象。
3	创建事例	通过拖放将原始对象从图形库复制到设备画面中。 通过链接原型与结构的动态向导来连接此对象与(过程)变量： 通过用实际变量连接替换每个属性的原始变量连接，向导自动将所有必需的变量的结构组件与原始画面模块的正确属性相链接。 现在就有了一个在运行期间用当前变量值更新的对象。

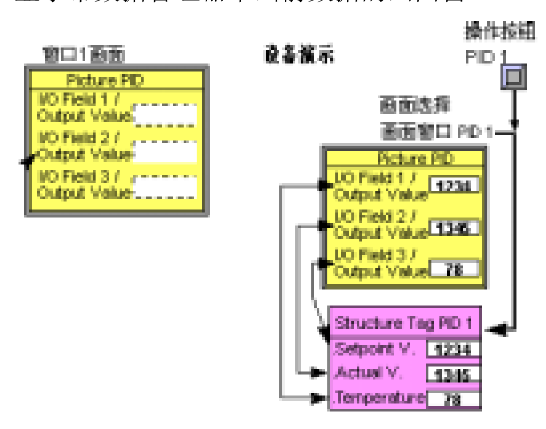
当使用动态的自定义对象和原型时，项目工程师必须确保对象上已经存储了必需的 C 动作。这些 C 动作不得删除，因为删除 C 动作会丢失模块的全部功能。

3.3.11.5 原始画面

原始画面技术再深入一步。当使用原型时，可以非常灵活地构造概念，以致于在调整创建的对象之后将自动更改原型。这种灵活性需要链接非常灵活的 C 脚本。原始画面技术通过使用可以反复集成到一个或更多母画面中的模板画面来付诸实现。模板画面只是一个模板，它只有集成到实际对象中之后才会“起作用”。基于模板(=原始画面)的对象通过“事例”来实现。对于一个模板，可以创建多个事例(即实际对象)。

画面窗口模板

显示带数据管理器中当前数据的画面窗口。



在(模板内)最重要的位置处随后进行更改，并且影响所有的应用程序(事例)。这样可以使其成为避免对不同位置进行繁琐的拖动更改的有效程序。

在一个母画面中最多可以显示 30 个特定模板类型的事例(也就是对象)。如果使用不同的原型，则可以使用 30 个以上的对象。

原始画面是创建后复制到库中的画面模块，因此它们可以反复使用。所用的画面模块在设备画面中作为模板的事例来使用。例如，这些复制品显示在模块中可视化的控制器或电机的当前数据。相应的控制器或电机组件自动显示。

不仅可以创建简单的画面模块，而且还可以创建复杂的画面模块。画面模块可以由若干个部分重叠或完全重叠但却组成单个共同单元的组件组成。例如有关电机的所有数据(当前状态的视图、进展数据和维护数据等)可以组合在一个对象中，并且仅在需要时进行更新。如果具有许多相同类型的电机，则只要创建画面模块一次，然后不断复制它就可以了。所有其它事都自动执行。

必须按照下列步骤来创建画面模块：

步骤	类型	组态
1	数据结构	通过变量管理器定义画面模块中要使用的数据结构。
2	画面窗口组合	在*图形编辑器*中组态画面模块的内容，例如棒图、I/O 域等。
3	链接数据结构与画面窗口组合	链接画面组件和带特殊 C 脚本(来自实例项目)的各数据结构组件。
4	画面窗口	将画面模块链接到画面窗口对象。
5	图形库	将画面窗口复制到图形库中。

现在通过从图形库中获取画面模块以便在设备画面中使用它。
首先看一下如何创建模板画面。

步骤	组态
1	在数据管理器中创建变量结构(结构数据类型)；在其中可以定义组成结构的变量数(成员变量)、变量的名称及其数据类型(BIT、SHORT 等)。例如带结构组件设定点值、实际值和温度的 PID。
2	创建要用作画面模块的画面。它通常小于监控画面的大小，并且尺寸上能做相应地调整。以其自身的能力所创建的每个画面可用于创建模板。 使用图形编辑功能来创建画面，并且在画面上定位图形变量(例如 I/O 域、棒图等)，但是不与变量链接。 在此画面中组态图形对象之间的内部关系(直接链接)，例如 I/O 域的输出值至棒图的变化受控制的传送。
3	现在将图形域链接到相应变量结构的结构组件。此链接组态是链接类型级的变量(仅指模板)，而不是具体的过程对象。 为此，在 WinCC 光盘(\Samples)上可以找到预备的实例项目。 在此项目的项目库(\Template)中执行此连接的是一个名为 TemplateInit 的自定义对象。它位于图形库中，并且可以从图形库拖放到要标准化的画面中。 *TemplateInit* 已经具有完整的脚本逻辑。此逻辑使用 *ConnectionTable*，它在组态期间作为表格填入并且准确地包含上面引用的带有下划线的条目。以上就是用于定义属性和结构组件之间链接的方法。 这些链接的组合可以在模板内设置，也可以从外部设置。为此，特定的项目图形库包含自定义的对象，它们看似简单的按钮，但实际上包含有关要调用的模板的参数化信息。 用于实现准备生成的原型的所有脚本必须复制到目标项目。参见章节末尾步骤 8 - 10 下最后几步的描述。若没有这些项目函数，就不能实现这些原型。
4	此画面要用作过程框。 为此，在临时画面(也就是说只是中间步骤需要它)中创建一个画面窗口对象，并且将此对象的画面名称属性与包含画面模块的画面相连。
5	通过拖放将画面窗口对象复制到图形库中。

现在该模板可以在设备画面中反复使用。当输入名称之后，自动与过程变量进行链接。

步骤	类型	组态
6	定义变量	定义相应数据类型的(过程)变量(例如 PID 类型的 PID_1)。
7	创建事例	从图形库复制画面窗口模块。 将正在使用的(过程)变量名(例如 PID_1)分配给画面窗口 - 对象名：将*画面窗口对象* → *画面窗口* → *对象名* → *静态* D 设置为 PID_1

在画面中定位画面模块时，用已构造的(过程)变量名来命名，该变量的值包含过程对象的状态数据。然后在运行期间画面模块自动获取状态数据。
有许多 C 脚本用于这些作为项目函数存储的原始画面。为了能使用已准备好的 C 脚本，必须采用实例项目中的下列脚本。请按如下操作：

步骤	类型	组态
8	复制函数文件	在路径 \Library 下，将全部所需的函数 (.fct) (例如 LinkConnectionTable.fct)从实例项目复制到用户的项目文件夹 \Library 中。
9	在项目中公布	在 WinCC 中启动全局脚本编辑器(全局脚本 → R → 打开)，并用以下方法公布这些新函数： 通过单击再生成头文件按钮，可以向项目函数的函数目录公布新函数。在项目函数列表中现在可以看到新函数。
10	传送自定义的对象	通过项目库使自定义的对象变为可用。对于一个库内还未存储任何自己的符号的新项目，只要将库从实例项目复制到用户项目中： 将\Library\library.pxl → 复制到用户的路径\Library！ 另外也可以使用导出机制将自定义的对象从一个项目传送至另一个项目。 导出实例项目中期望的符号作为.emf 文件(文件 → 导出)，然后通过插入 → 导入将自己项目内的这些.emf 符号导入临时画面中。通过拖放将这些符号传送至用户项目库。也可以使用单独的文件夹，例如 Template。

画面模块(标准化的画面段或模板)有很好的节省作用。例如，它们被看作带变量参考的对象模型并存储在图形库中。它们可以从图形库中取出、在设备画面中定位并在运行时自动提供数据。因此不必再组态各变量与图形段(例如 I/O 域、棒图等)的链接。

当使用这些原始画面时，可以用不同的方法为画面模块组件提供当前名称。通过以下简要说明的不同方法来进行命名：

- 根据选择的画面窗口来确定事例名称：为此在画面窗口中的打开画面事件处存储预定义的项目函数(EnableTemplateInstance)。
- 通过*输入/输出变量*定义事例名称，该变量在画面窗口中通过确定事例名称的脚本自动读取：这通过使用预备的自定义对象 Instance Call Button + Template 来完成。
- *按钮*直接将事例名称传送至调用的画面窗口：这通过使用预备的自定义对象 *InstanceCallButtons+Template* 来完成。

WinCC 光盘(\Samples)上的实例项目中可以找到不同变量的详细实例。

3.3.11.6 OCX 对象

OCX 或 ActiveX 对象是可用作装载组件的画面模块。WinCC 提供了许多这种对象，例如 WinCC 数字/模拟时钟控制。
这些模块可以非常容易地集成到设备画面中。

步骤	类型	组态
1	插入 OCX 对象	在智能对象处*图形编辑器*的对象选项板中，选择条目 OLE 控制(OCX)。通过按住动作 将对象拖至设备画面中，然后从显示的对话框中选择期望的元素。
2	连接属性	插入的对象同样具有属性和事件。哪些属性和事件可用取决于指定的 OCX。 例如，与变量的连接可以在属性过程驱动程序连接处创建。

创建

OCX 画面模块必须用独立的开发环境来创建。它可以是 Microsoft Visual C++ 5 或者 Microsoft Visual Basic 5。
此方法用于修改和改进画面模块。OCX 模块的功能非常强大，但是不能用 WinCC 组态资源来创建。为此必须求助于外部方法来创建或修改。
与容易更改并且功能强大的 OCX 编程相反，原始画面技术完全可由单纯的 WinCC 资源来解决。也就是说不必具有任何 OCX 编程的知识。
如今已经有许多这样的模块可供使用。其中包括可用作 PCS7 面板的完整模块，这些面板与操作和观察两方面的集成以及设备区域中的 PLC 编程(PCS7)有关。

注册

创建或购买的 OCX 模块必须在相应的 WinCC 站上注册。通过查看图形编辑器的选择框，可以知道在 WinCC 站上可以使用哪些 OCX 对象(参见上面的描述)。在对话框中列出了计算机上注册的所有 OCX 元素。OCX 元素以扩展名为.OCX 或.dll 的文件形式存储在计算机上。
如果模块还未注册，则可以在 WinCC OLE 控制对话框中进行注册。此对话框包含一个用于注册和删除当前所选组件的注册的按钮。
WinCC 站上必须具有用于执行注册的相关文件。
OCX 组件的兼容性和功能必须由组态人员亲自进行测试。只有那些标有 WinCC 的 OCX 模块已经在 WinCC 环境中使用和测试。

3.3.12 在线组态(运行系统) - 注意事项和限制

有关在线组态必须注意许多要点。
由于各种原因，有少量的更改不能在在线模式下完成、只能在特定条件下才能完成或者过一段时间才能生效。

WinCC 资源管理器

下列更改没有采纳：

- 在计算机列表中更改计算机的类型

运行期间不可以进行下列组态步骤：

- 删除/重新命名变量
- 更改变量的数据类型

报警记录

下列更改没有采纳：

- 更改归档/报表
- 更改组消息
- 运行激活期间 500 条单个消息之后的每个消息

运行期间不可以进行下列组态步骤：

- 无限制

变量记录

下列更改没有采纳：

- 无限制。

运行期间不可以进行下列组态步骤：

- 用户归档的表格可以创建但不能更改
- 删除变量记录和用户归档中的数据

运行期间组态的例外情况：

- 变量记录的运行系统 API 可用于编辑和删除用户归档的表格

全局脚本

下列更改没有采纳：

- 对向导脚本的更改只有在重新启动图形编辑器之后才能采纳
- 修改的向导脚本

运行期间不可以进行下列组态步骤：

- 无限制

报表编辑器

下列更改没有采纳：

- 对消息顺序报表的更改，因为一旦启动，它在运行期间就始终保持激活状态并且不会重新装载布局信息

运行期间不可以进行下列组态步骤：

- 无限制

冗余

下列更改没有采纳：

- 伙伴的计算机名不能传送至第三台计算机
- 自动切换器不能更改，也就是说用户起初必须组态自动切换器要切换到什么位置。但是如果另一个发生故障，它还是会切换回来

运行期间不可以进行下列组态步骤：

- 无限制

SIMATIC S7 Protocol Suite 或 S/7PMC 通道

下列更改没有采纳：

- 未在线采纳与 S7Chn.ini (未公开)相关的诊断参数
- 虽然在线采纳了对通讯地址的所有更改，但是它们只有在建立连接后才能判断

运行期间不可以进行下列组态步骤：

- 无限制

文本库

下列更改没有采纳：

- 无限制。
- 在文本库中，通过文件 → 将*更改发送至激活的项目*来采纳更改的*文本*
- 在报警记录中，更改通过文件 → 保存复制到文本库中

运行期间不可以进行下列组态步骤：

- 无限制

用户管理器

下列更改没有采纳：

- 对用户授权的更改只有再次退出/登录之后才能生效

运行期间不可以进行下列组态步骤：

- 无限制

4 WinCC C 课程

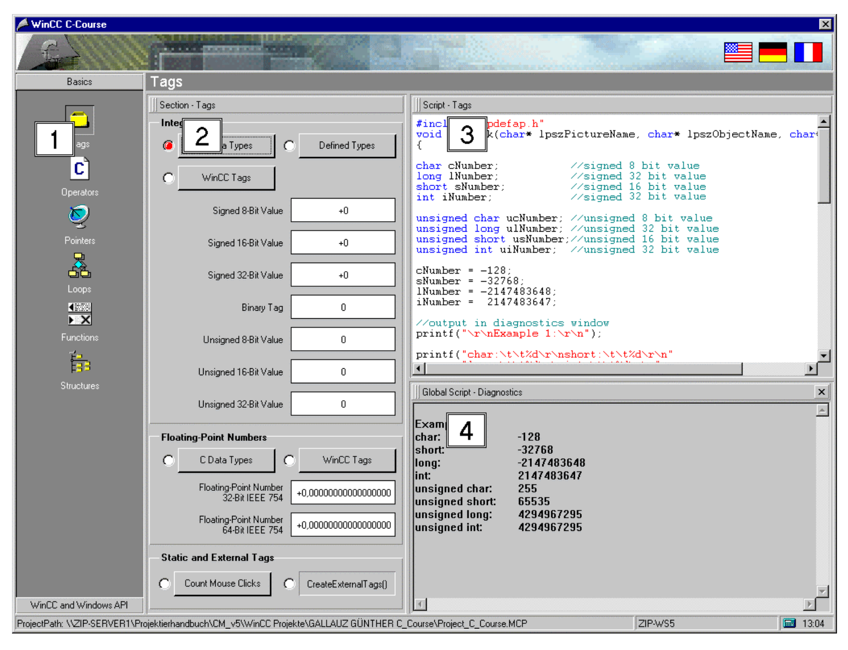
要使对象动态化，在 WinCC 中有多种不同的选项可用。其中包括变量连接、动态对话框和直接连接。通过它们可以实现复杂的动态。然而，随着要求的增加它们会有限制。对于用 户来说，组态 C 动作、项目函数或动作可以有更广的范围。它们在 WinCC 脚本语言 C 中创建。对于许多应用来说，不必具有非常全面的 C 语言知识。它足以为现有函数提供参数。然而，为了使用 WinCC 脚本语言 C 的全部功能，需要具备有关这种编程语言的基本知识。本课程可以为用户提供这些知识。

目标组

本课程用来为不熟悉 C 语言的人员提供有关编程语言 C 的常规应用的基本知识。具有 C 语言编程经验的编程员可以学习 C 语言应用于 WinCC 时的特性。
本课程的实例项目可以直接从在线文档复制到用户的硬盘驱动器中。缺省情况下，它将存储在 C:\Configuration_Manual 文件夹中。

 Project_C_Course

实例项目



实例项目的界面分成若干个部分。它们列于下：

- 浏览栏(1)：浏览栏允许选择与不同章节相关的图片。
- 章节窗口(2)：章节窗口显示分配给各章节的画面。这些画面包含特定章节中描述的所有实例。大多数实例都以按钮的形式创建。
- 脚本窗口(3)：脚本窗口显示章节窗口中当前所选实例的代码。当前所选择的实例在章节窗口中用红色标记。
- 诊断窗口(4)：诊断窗口显示通过 printf()函数启动的各种实例的所有输出。

4.1 C 脚本的开发环境

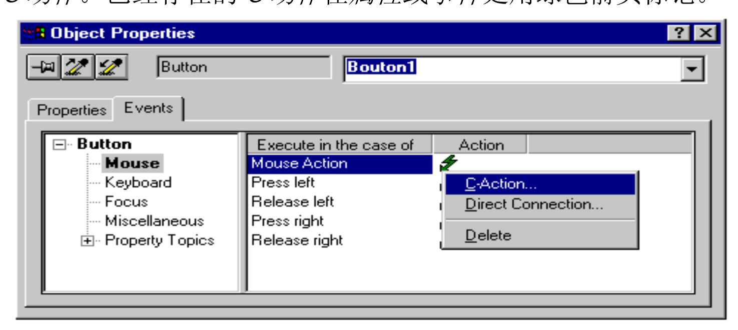
对于 C 脚本的创建，WinCC 提供两个不同的编辑器。一个是图形编辑器中的动作编辑器，用于在对象处创建 C 动作；另一个是全局脚本编辑器，用于创建项目函数和全局动作。脚本语言的语法与采用 ANSI 的标准 C 语言相一致。
在 WinCC 中编程语言 C 的另一个应用领域是关于动态向导的创建。为此，可以使用一个单独的编辑器。此编辑器的应用在有关动态向导的实例中解释，不会在常规概述中进行讨论。

4.1.1 图形编辑器的动作编辑器

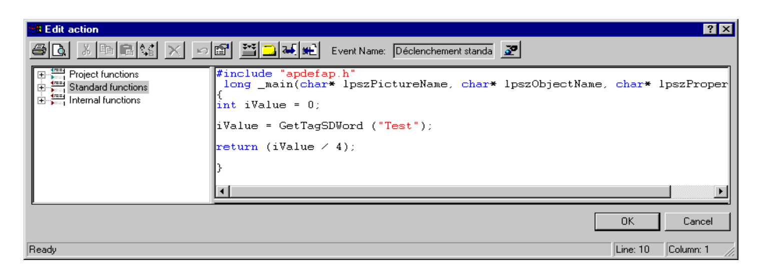
在图形编辑器中，可以通过 C 动作使对象属性动态化。同样，也可以使用 C 动作来响应对象事件。

动作编辑器

对于 *C 动作*的组态，可以使用动作编辑器。此编辑器可以在*对象属性*对话框中通过以下方法打开，即 R 期望的属性或事件，然后从显示的弹出式菜单中选择 *C 动作*。已经存在的 *C 动作*在*属性*或*事件*处用绿色箭头标记。



在动作编辑器中，可以编写 *C 动作*。对于属性的 *C 动作*，必须定义触发器。对于事件的 *C 动作*，由于事件本身就是触发器所以不必再定义。完成的 C 动作必须进行编译。如果编译程序没有检测到错误，则可以通过单击*确定*退出动作编辑器。



C 动作的结构

通常，一个 *C 动作*相当于 C 中的一个函数。*C 动作*有两种不同的类型：为*属性*创建的动作和为*事件*创建的动作。通常，属性的 C 动作用于根据不同的环境条件控制此属性的值(例如变量的值)。对于这种类型的 *C 动作*，必须定义触发器来控制其执行。*事件*的 *C 动作*用来响应此*事件*。

属性的 C 动作

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{

/*1*/ long lReturnValue;

/*2*/ lReturnValue = GetTagSDword("S32i_course_test_1");

/*3*/ return lReturnValue;

}
```

上述实例代码代表一个典型的属性的 C 动作。各部分的含义描述如下。

- **标题(灰色)**：灰色阴影显示的前三行构成 *C 动作*的标题。该标题自动生成并且不能更改。除返回值类型(在实例代码中为 long)之外，所有属性的函数标题都完全相同。将三个参数传送给 *C 动作*。它们是画面名称*(lpsz Picture Name)*，对象名*(lpszObjectName)*和属性名*(lpszPropertyName)*。
- **变量声明(1)**：在可以编辑的第一个代码段中声明使用的变量。在本实例代码中，指的是一个 *long* 型的变量。
- **数值计算(2)**：在本段中，执行属性值的计算。在实例代码中，只读入一个 WinCC 变量的数值。
- **数值返回(3)**：将计算得出的属性值赋给属性。这通过 *return* 命令来完成。

事件的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{

/*1*/ long lValue;

/*2*/ lValue = GetTagSDWord("S32i_course_test_1");

      SetLeft(lpszPictureName,lpszObjectName,lValue);

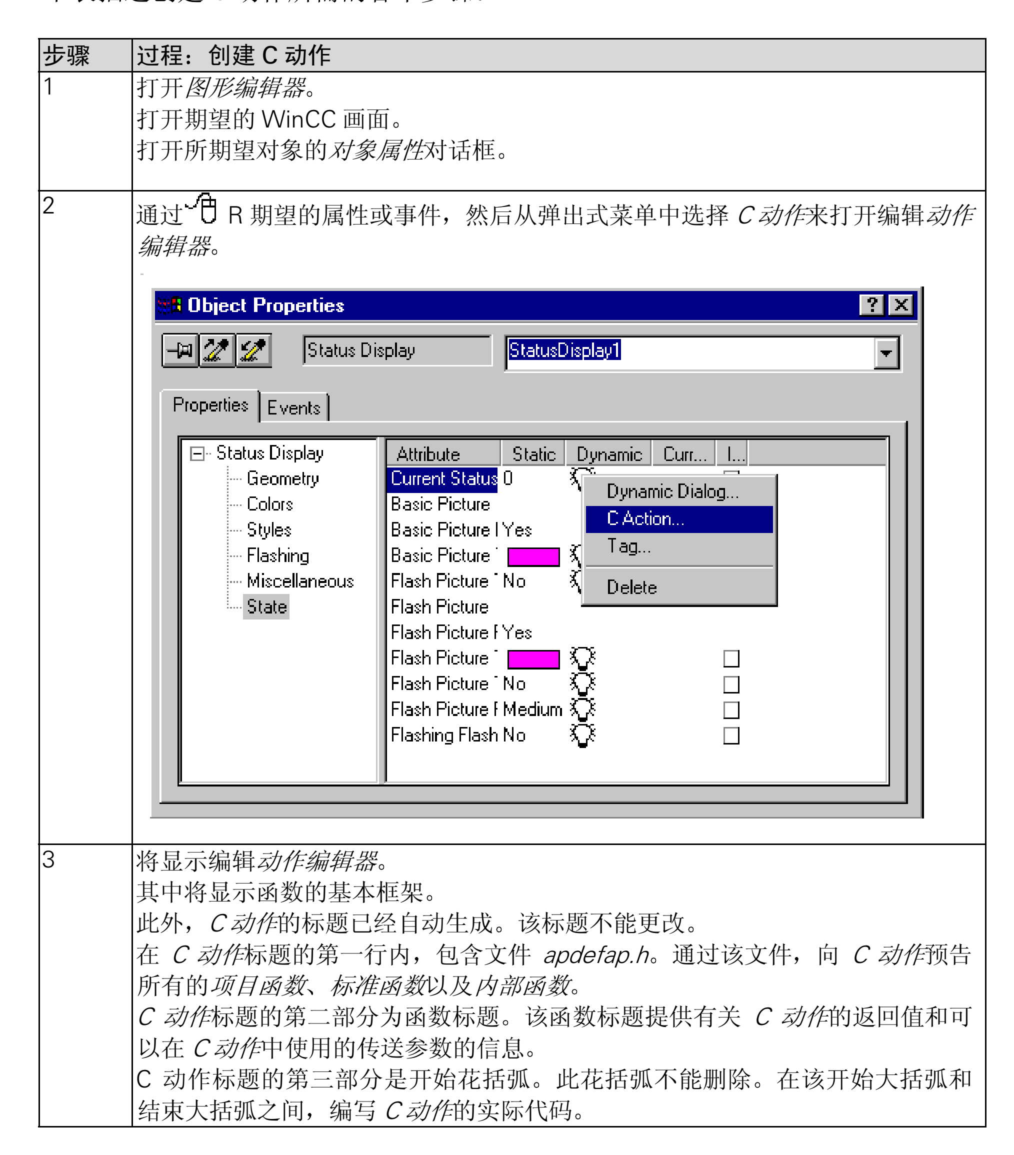
}
```

上述实例代码代表一个典型的事件的 *C 动作*。各部分的含义描述如下。

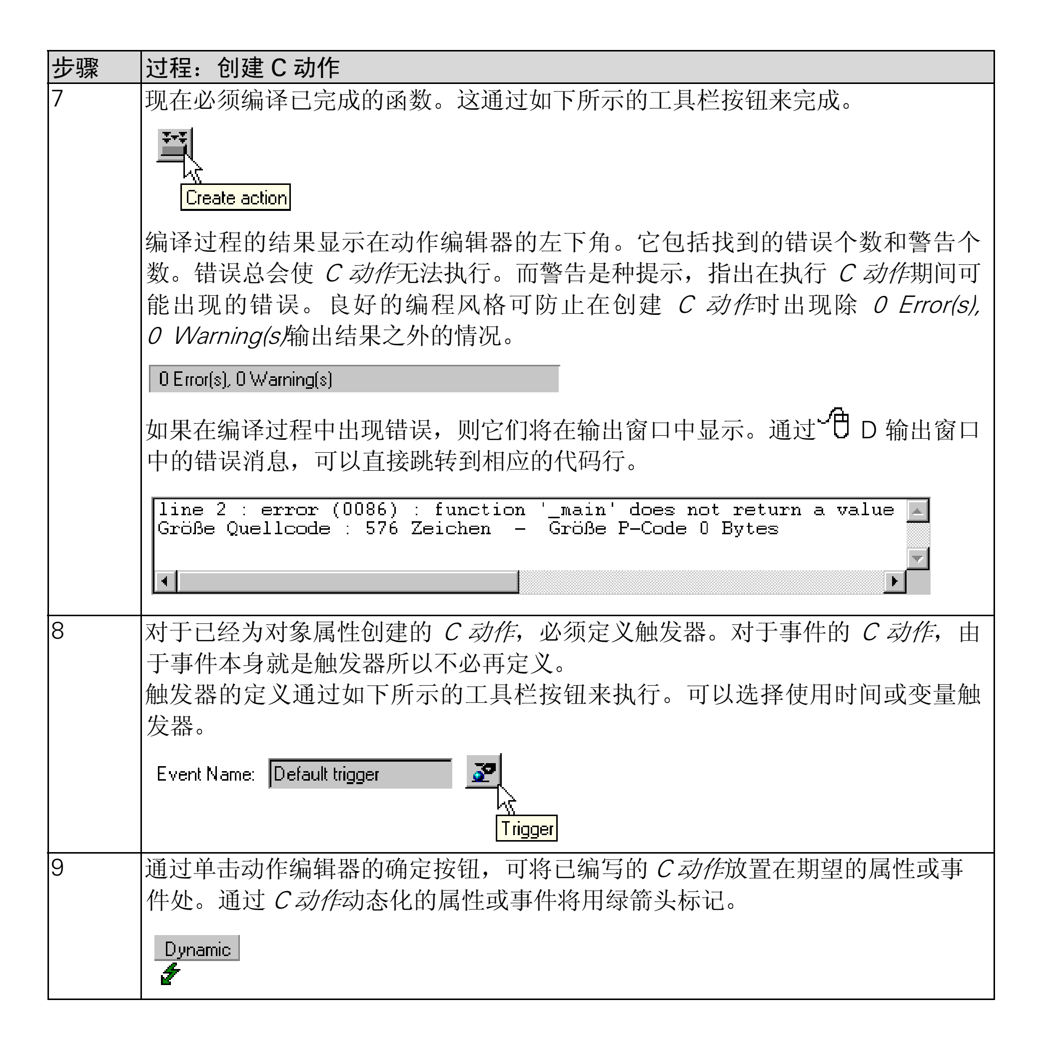
- **标题(灰色)**：灰色阴影显示的前三行构成 *C 动作*的标题。该标题自动生成并且不能更改。对于不同类型的事件，其函数标题也不同。将参数 *lpszPictureName*(画面名称)、*lpszObjectName*(对象名)和 *lpszPropertyName*(属性名)传送给 *C 动作*。参数 *lpszPropertyName* 只包含与响应属性变化的事件相关的信息。可以传送附加的事件指定的参数。
- **变量声明(1)**：在可以编辑的第一个代码段中声明使用的变量。在本实例代码中，指的是一个 *long* 型的变量。
- **事件处理(2)**：在本段中，执行响应相应事件的动作。在本实例代码中，读入一个 WinCC 变量的数值。该数值作为位置 X 分配给自己的对象。事件的 *C 动作*的返回值为 *void* 类型，也就是说不需要返回值。

C 动作的创建

下表描述创建 C 动作所需的各个步骤。

步骤	过程：创建 C 动作
1	打开*图形编辑器*。 打开期望的 WinCC 画面。 打开所期望对象的*对象属性*对话框。
2	通过 R 期望的属性或事件，然后从弹出式菜单中选择 *C 动作*来打开编辑*动作编辑器*。 
3	将显示编辑*动作编辑器*。 其中将显示函数的基本框架。 此外，*C 动作*的标题已经自动生成。该标题不能更改。 在 *C 动作*标题的第一行内，包含文件 *apdefap.h*。通过该文件，向 *C 动作*预告所有的*项目函数*、*标准函数*以及*内部函数*。 *C 动作*标题的第二部分为函数标题。该函数标题提供有关 *C 动作*的返回值和可以在 *C 动作*中使用的传送参数的信息。 C 动作标题的第三部分是开始花括弧。此花括弧不能删除。在该开始大括弧和结束大括弧之间，编写 *C 动作*的实际代码。

步骤	过程：创建 C 动作
4	其它自动生成的代码部分包括两个注释块。若要使*交叉索引*编辑器可以访问 *C 动作*的内部信息，则需要这些块。要允许 C 动作中语句重新排列也需要这两个块。如果这些选项都不用，则也可以删除这些注释。 第一个注释块用于定义 *C 动作*中所使用的 WinCC 变量。在程序代码中，必须使用所定义的变量名称而不是实际的变量名称。 第二个注释块用于定义 C 动作中使用的 WinCC 画面。在程序代码中也必须使用定义的画面名称而不是实际的画面名称。 有关该主题的实例代码可在此表格的后面找到。它包含 WinCC 变量和 WinCC 画面的定义以及这些定义的应用。
5	编写执行期望计算的函数主体、动作等。 有多种编程辅助工具可供使用。其中一种辅助工具是变量选择对话框。此对话框通过如下所示的工具栏按钮打开。在显示的*选择变量*对话框中，选择 WinCC 变量然后单击确定来确认。于是将在 *C 动作*中当前光标位置处插入所选 WinCC 变量的名称。 Variable selection
6	另一种辅助工具是动作编辑器左窗口中的函数选择。利用函数选择，可以在 C 动作中的当前光标位置处自动插入所有可用的*项目函数*、*标准函数和内部函数*。 Project functions Standard functions Internal functions 为此，通过 D 来选择期望的函数。将显示*指定参数*对话框，它包含所有必须输入的参数及其数据类型的列表。该函数可以在*数值*列中进行参数化。除简单的文本输入之外，*选择变量*、*图形对象*和*画面*选项都可用。 为了在 *C 动作*中的当前光标位置处插入函数，通过单击*确定*来确认对话框。 Assigning Parameters Function Name SetTagSDWord Description / Data Type / Value Tag_Name / Tag / Tag_Name value / long int / val Variable selection Graphic objects Pictures OK Cancel



步骤	过程：创建 C 动作

7	现在必须编译已完成的函数。这通过如下所示的工具栏按钮来完成。
	Create action
	编译过程的结果显示在动作编辑器的左下角。它包括找到的错误个数和警告个数。错误总会使 *C 动作*无法执行。而警告是种提示，指出在执行 *C 动作*期间可能出现的错误。良好的编程风格可防止在创建 *C 动作*时出现除 *0 Error(s), 0 Warning(s)*输出结果之外的情况。
	0 Error(s), 0 Warning(s)
	如果在编译过程中出现错误，则它们将在输出窗口中显示。通过 D 输出窗口中的错误消息，可以直接跳转到相应的代码行。
	`line 2 : error (0086) : function '_main' does not return a value`
	`Größe Quellcode : 576 Zeichen  –  Größe P-Code 0 Bytes`

8	对于已经为对象属性创建的 *C 动作*，必须定义触发器。对于事件的 *C 动作*，由于事件本身就是触发器所以不必再定义。 触发器的定义通过如下所示的工具栏按钮来执行。可以选择使用时间或变量触发器。 Event Name: Default trigger Trigger
9	通过单击动作编辑器的确定按钮，可将已编写的 *C 动作*放置在期望的属性或事件处。通过 *C 动作*动态化的属性或事件将用绿箭头标记。 Dynamic

WinCC 标签定义和画面定义

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{
// WINCC:TAGNAME_SECTION_START
// syntax: #define TagNameInAction "DMTagName"
#define S32I_COURSE_TEST_1 "S32i_course_test_1"
// next TagID : 1
// WINCC:TAGNAME_SECTION_END

// WINCC:PICNAME_SECTION_START
// syntax: #define PicNameInAction "PictureName"
#define CC_0_STARTPICTURE_00 "cc_0_startpicture_00.Pdl"
// next PicID : 1
// WINCC:PICNAME_SECTION_END

SetTagSDWord(S32I_COURSE_TEST_1,100);

OpenPicture(CC_0_STARTPICTURE_00);

return 0;

}
```

如果创建新的 *C 动作*，则自动生成的代码将包括两个注释块。若要使*交叉索引*编辑器可以访问 *C 动作*的内部信息，则需要这些注释块。要允许 *C 动作*中语句重新排列也需要这两个块。

- **变量定义**：第一个注释块用于定义 *C 动作*中使用的 WinCC 变量。该注释块以 *//WINCC: TAGNAME_SECTION_START* 作为开始，以 *//WINCC: TAGNAME_SECTION_END* 作为结束。在这两行之间，定义 *C 动作*中使用的所有 WinCC 变量的名称。通过预处理程序命令*#define* 后跟定义的名称(在本实例代码中为 *S32I_COURSE_TEST_1*)，其后再接 WinCC 变量的名称(在本实例代码中为 *S32i_course_test_1*)来进行定义。
- **画面定义**：第二个注释块用于定义 *C 动作*中使用的 WinCC 画面。该注释块以行 *//WINCC: PICNAME_SECTION_START* 作为开始，以行 *//WINCC: PICNAME_SECTION_END* 作为结束。在这两行之间，定义 *C 动作*中使用的所有 WinCC 画面的名称。它遵循的规则与上面所描述的定义变量名称时所遵循的规则相同。
- **应用**：在实际程序代码中，必须使用定义的值而不是实际的变量和画面名称。在编译 *C 动作*之前，预处理程序将用实际名称替换所有定义的名称。

4.1.2 全局脚本编辑器

全局脚本编辑器用于创建项目函数、标准函数和动作。

项目函数

如果在 C 动作中经常需要相同的功能，则该功能可以在项目函数中公式化。在 WinCC 项目的所有 C 动作中都可以按照调用所有其它函数一样的方式来调用项目函数。下面列出了使用项目函数相对于在 C 动作中创建完整的程序代码的优点：

- **编辑的中心位置**：*项目函数*的改变会影响所有正在使用该函数的 C 动作。如果没有使用项目函数，则必须手动修改所有相关的 C 动作。这不但可以简化组态，而且可以简化维护和故障检测工作。
- **可重用性**：一旦一个项目函数编写完并进行了广泛的测试，则它随时都可以再次使用，无需附加的组态或新的测试。
- **画面容量减少**：如果并不是在对象的 *C 动作*中直接放置完整的程序代码，则画面容量将减少。这可以使画面打开的速度更快并且在运行系统中性能更佳。
- **口令保护**：*项目函数*可以通过指定口令进行保护，以防更改。这样可以保护组态数据以及用户的技术决窍。

项目函数只能在项目内使用。它们存储在 WinCCProjectFolder\LIBRARY 文件夹内，并在相同文件夹中的 ap_pbib.h 文件内定义。

许多*标准函数*都可以使用。与*项目函数*相反，标准函数可用于所有的 WinCC 项目。可以更改现有的*标准函数*。也可以创建新的*标准函数*。

标准函数与项目函数的区别仅在于它们的可用性：标准函数可以跨项目使用，然而项目函数只能在项目内使用。标准函数存储在 WinCCInstallationFolder\APLIB 文件夹内，并在同一文件夹中的 ap_glob.h 文件内定义。

内部函数

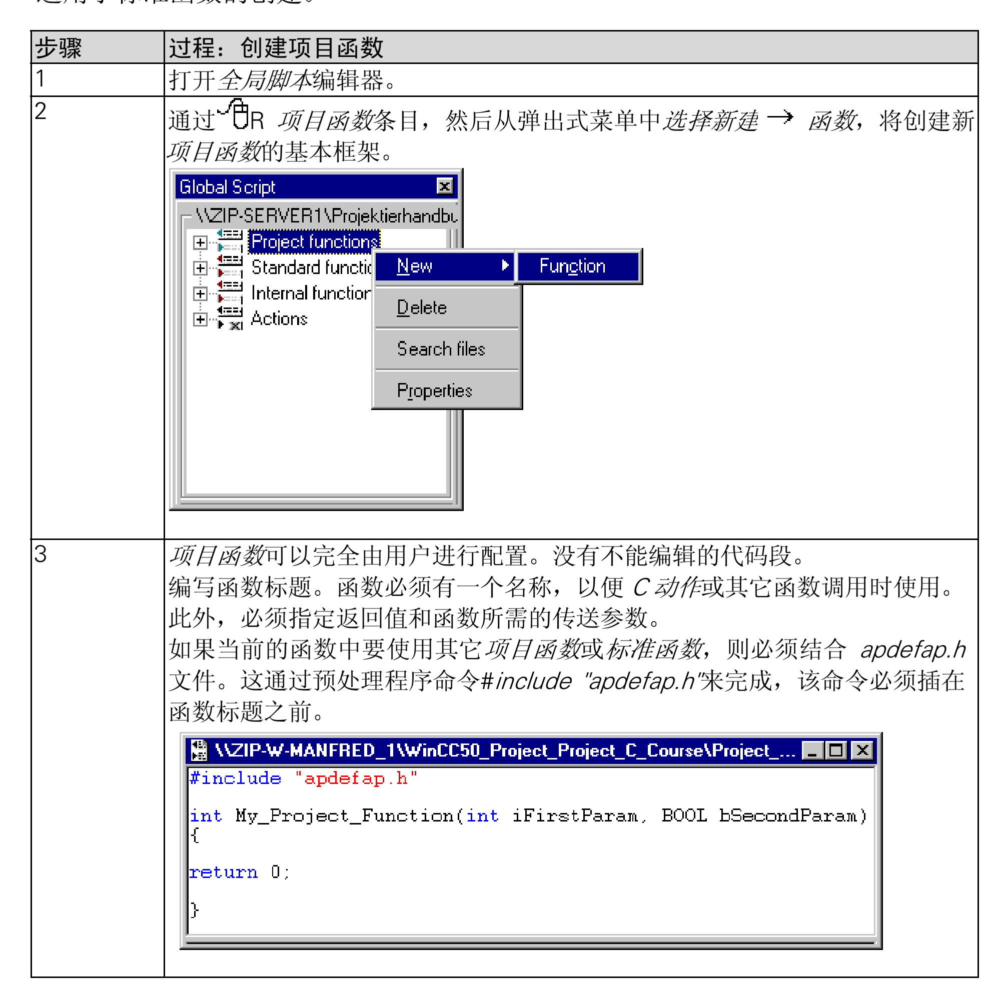
除项目函数和标准函数之外，还有内部函数。其中，它们是标准的 C 函数。用户不能对其进行更改，也不能创建新的内部函数。

动作

动作(与先前描述的函数相反)不能由 C 动作或其它函数调用。必须为动作指定触发器来控制其执行。它在运行系统中执行时与当前所选择的画面无关。
可以组态全局动作，即跨项目动作。在这种情况下，它们存储在 WinCCProjectFolder\PAS 文件夹中。也可以组态局部动作(指定计算机的动作)，它们将存储在 WinCCProjectFolder\ComputerName\PAS 文件夹中。
如果在计算机的启动列表中选中了*全局脚本运行系统*，则一旦项目启动，属于该计算机的所有*全局动作*和所有*局部动作*都将被激活。

创建项目函数

创建项目函数所需的步骤与创建标准函数所需的步骤完全相同。因此以下说明也适用于标准函数的创建。



步骤	过程：创建项目函数
1	打开*全局脚本*编辑器。
2	通过 R *项目函数*条目，然后从弹出式菜单中*选择新建* → *函数*，将创建新*项目函数*的基本框架。
3	*项目函数*可以完全由用户进行配置。没有不能编辑的代码段。 编写函数标题。函数必须有一个名称，以便 *C 动作*或其它函数调用时使用。此外，必须指定返回值和函数所需的传送参数。 如果当前的函数中要使用其它*项目函数*或*标准函数*，则必须结合 *apdefap.h* 文件。这通过预处理程序命令*#include "apdefap.h"*来完成，该命令必须插在函数标题之前。


```
#include "apdefap.h"

int My_Project_Function(int iFirstParam, BOOL bSecondParam)
{

return 0;

}
```

步骤	过程：创建项目函数
4	编写函数主体。 为此，可以使用与编写 *C 动作*相同的辅助工具。特别是变量选择和函数选择。

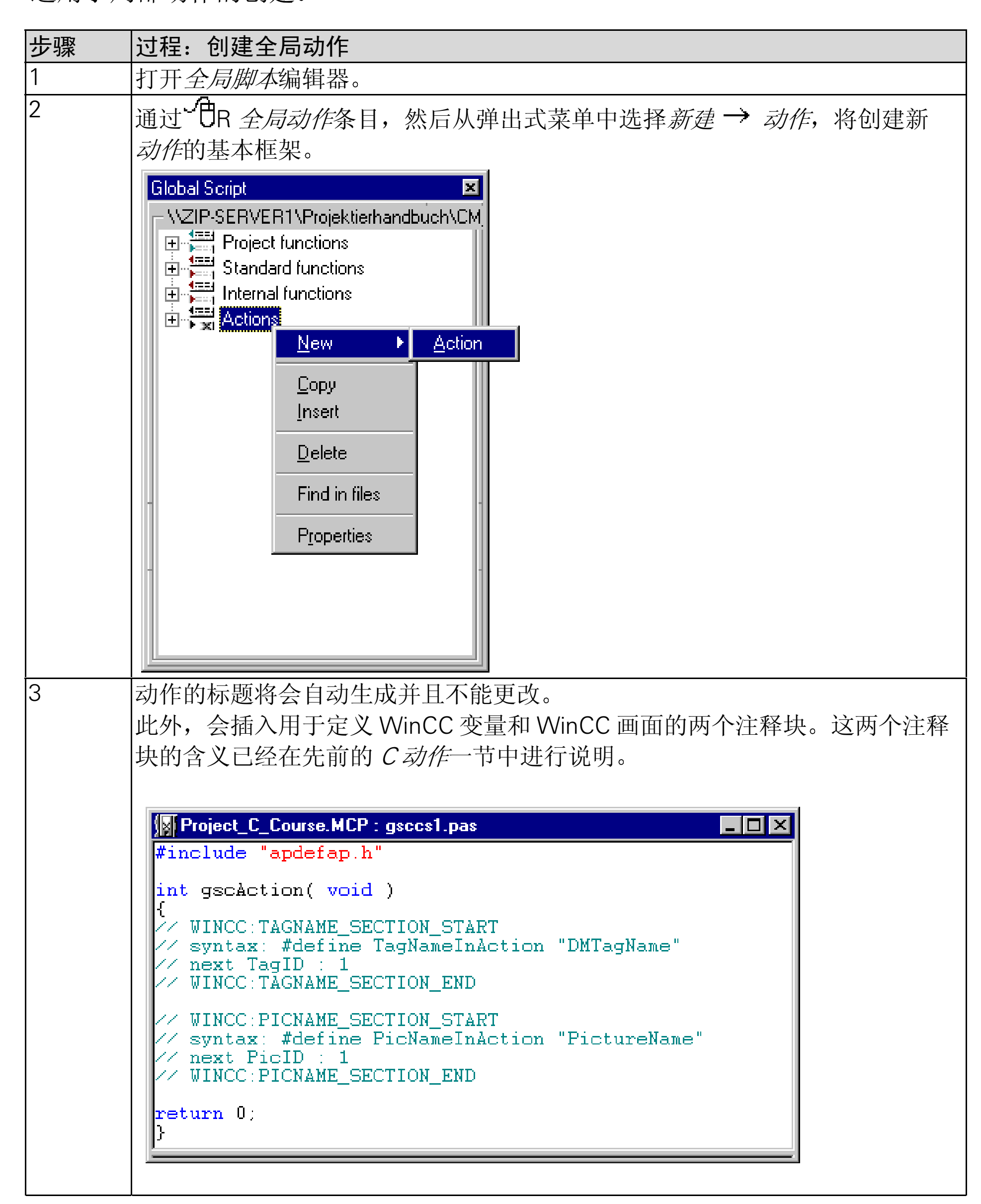
5	已完成的函数现在必须进行编译。这通过如下所示的工具栏按钮来完成。
	Create action
	编译过程的结果显示在输出窗口中。将列出产生的错误和警告，并且显示其数量。通过D 输出窗口中的错误消息，可以直接跳转到相应的代码行。
	`Compiling...`
	`line 3 : error (0086) : function 'My_Project_Function' does not return a va`
	`1  Error(s),0  Warning(s)`

6	通过如下所示的工具栏按钮，可以将描述添加到*项目函数*中。可以与描述一起定义一个口令，以保护项目函数免遭未授权人员的访问。 Info / Trigger
7	完成的*项目函数*必须用合适的名称进行保存。

创建全局动作

创建全局动作所需的步骤与创建局部动作所需的步骤完全相同。因此以下说明也适用于局部动作的创建。

步骤	过程：创建全局动作
1	打开*全局脚本*编辑器。
2	通过 R *全局动作*条目，然后从弹出式菜单中选择*新建* → *动作*，将创建新*动作*的基本框架。
3	动作的标题将会自动生成并且不能更改。 此外，会插入用于定义 WinCC 变量和 WinCC 画面的两个注释块。这两个注释块的含义已经在先前的 *C 动作*一节中进行说明。



步骤	过程：创建全局动作
4	编写动作主体。 为此，可以使用与编写 *C 动作*相同的辅助工具。特别是变量选择和函数选择。 *动作*具有 *int* 类型的返回值。然而，该返回值不能由用户计算。缺省情况下，返回数值 *0*。
5	通过如下所示的工具栏按钮，可以如同函数描述一样将描述添加到*动作*中。也可以定义口令来保护*项目函数*免遭未授权人员的访问。 与函数相比，它还需要设置一个触发器来控制*动作*的执行。对于*动作*触发器的选择，用户所具有的选择范围要比对象的 *C 动作*触发器的选择范围大。其中，可以编写一次执行过程。 Info / Trigger
6	完成的*动作*必须进行保存。

测试输出

随后可以执行程序来测试输出。这样便于在开发期间进行故障检测和错误诊断。测试输出可以通过 printf()函数来启动。通过该函数，不但可输出简单文本，而且可输出当前变量值。为了使输出文本可见，必须组态全局脚本诊断窗口。

printf()函数

printf()函数允许执行测试输出功能。该函数的实例应用显示如下：

```
printf("I am %d years old\r\n",iAge);
```

printf()函数至少需要一个参数。该参数是一个字符串。要传送的附加参数的类型和数量取决于该字符串。
字符%由 *printf()*函数用作要在该位置插入变量值的标识符。跟在字符%之后的字符确定该变量的数据类型。上例中所使用的字符组合%d 表明输出为十进制数。其它可能的组合及其描述可参见下表：

参数	描述
%d	输出十进制数(int 或 char)
%ld	长整型变量作为十进制数输出
%c	输出字符(char)
%x	以十六进制格式输出数值(用小写字母 a...f)
%X	以十六进制格式输出数值(用大写字母 A...F)
%o	以八进制格式输出数值
%u	输出十进制数(专用于 unsigned 类型)
%f	以浮点数计数制输出浮点型数值，例如 3.43234
%e	以指数计数制输出浮点型数值，例如 23e+432
%E	同%e，但使用大写的 E，例如 23E+432
%s	输出字符串(char*)
%le	输出双精度型数值
%%	输出%字符
\n	换行输出(回车)
\r	进一行输出
\t	制表位输出
\\	输出\字符

全局脚本诊断窗口

由 printf()函数指定的文本输出显示在全局脚本诊断窗口中。组态这种诊断窗口的步骤在下表内描述：

步骤	过程：全局脚本诊断窗口
1	打开*图形编辑器*。 打开期望的 WinCC 画面。
2	组态*智能对象* → *应用窗口*。 将应用窗口置于画面内之后，将打开窗口*内容*对话框。从列表框中选择*全局脚本*条目。通过单击*确定*退出对话框。 将打开*模板*对话框。从列表框中选择 *GSC 诊断*条目。同样通过单击*确定*退出对话框。
3	为了便于利用*全局脚本诊断*窗口，建议将*对象属性*对话框内*其它*条目下的所有属性设置为*是*。



步骤	过程：全局脚本诊断窗口
4	如果项目在运行，则由 *printf()*函数指定的文本输出将显示在诊断窗口中。如果用工具栏上相应的按钮中止更新，则可以保存或打印输出窗口内容。

4.2 变量

在 WinCC 项目 Project_C_Course 中，有关变量主题的实例可以通过单击如下所示的浏览栏图标来访问。实例在 cc_9_example_00.PDL 画面中组态。



Tags

变量

变量是由程序处理的数据对象。变量只有在定义之后才能使用。在第一条指令可以执行之前，必须先定义程序中使用的所有变量。
变量可以比作一个容器。通过变量名，我们给容器一个唯一的名称。容器中内容的类型通过其数据类型来指定。容器的初始内容通过其初始值来指定。在大多数情况下，该内容将在程序执行过程中进行处理。
此处所描述的变量不应误认为是 WinCC 变量。它们只能在程序代码中使用。
以下程序代码说明了定义变量的一个实例。在该实例中，用名称 iNumber 来定义一个 int 数据类型的变量。代码行以分号结束。变量名的前面是描述数据类型的前缀。这并非必须遵循，但它却使得在程序创建期间能够立即识别变量的数据类型。

```
int iNumber;
```

在定义变量时，也可以将其初始化。

```
int iNumber = 0;
```

常量

除变量之外，程序中也使用常量。它只是数值的直接应用。为了说明这种数值的含义，可以使用#define 命令为它定义符号常量。
以下程序代码说明了定义符号常量的一个实例。在该实例中，用数值 2147483647 来定义符号常量 MAX_INT_VALUE。注意代码行不得以分号结束。用大写字母表示符号常量是一般的编程规则，以便易于与变量区别。

```
#define MAX_INT_VALUE 2147483647
```

数据类型

C 所识别的基本数据类型列于下表。

数据类型	描述
char	一个字节，可以接受一个字符
int	整型数值
float	单精度型浮点数
double	双精度型浮点数

char 数据类型的变量需要一个字节的存储空间。其内容可以解释为一个字符或一个数字。
int 数据类型之前可以加关键字 signed 或 unsigned。关键字 signed 代表有符号数，关键字 unsigned 代表无符号数。
int 数据类型之前也可以加关键字 long 或 short。这些关键字也可以不带 int 而单独使用，其含义仍然相同。short (或 short int)数据类型的变量需要 2 个字节的存储空间，long (或 long int)数据类型的变量与 int 数据类型的变量一样需要 4 个字节的存储空间。
double 数据类型与 float 数据类型的区别仅在于其数值范围。用 double 数据类型，数字能以更高的精度表示。float 数据类型的变量需要 4 个字节的存储空间，然而 double 数据类型的变量需要 8 个字节的存储空间。

各数据类型的数值范围

每种数据类型都可以显示某一数值范围内的数值。区别在于不同的数据类型所需的存储空间不同，以及是有符号还是无符号数据类型。

数据类型	数值范围
int	-2 147 483 648 至 2 147 483 647
unsigned int	0 至 4 294 967 295
short	-32 768 至 32 767
unsigned short	0 至 65 535
long	-2 147 483 648 至 2 147 483 647
unsigned long	0 至 4 294 967 295
char	-128 至 127 (所有的 ASCII 字符)
unsigned char	0 至 255 (所有的 ASCII 字符)
float	-10 ^38 至 0 ^38
double	-10 ^308 至 0 ^308

4.2.1 实例 1 - C 的数据类型(整数)

在本实例中，用可用的 C 的缺省数据类型来显示整数。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 1* 组态了本实例。

C Data Types

按钮 1 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{

char cNumber;            //signed 8 bit value
long lNumber;            //signed 32 bit value
short sNumber;           //signed 16 bit value
int iNumber;             //signed 32 bit value

unsigned char ucNumber;  //unsigned 8 bit value
unsigned long ulNumber;  //unsigned 32 bit value
unsigned short usNumber;//unsigned 16 bit value
unsigned int uiNumber;   //unsigned 32 bit value

cNumber = -128;
sNumber = -32768;
lNumber = -2147483648;
iNumber =  2147483647;

//output in diagnostics window
printf("\r\nExample 1:\r\n");

printf("char:\t\t%d\r\nshort:\t\t%d\r\n"
       "long:\t\t%d\r\nint:\t\t%d\r\n",
       cNumber,sNumber,lNumber,iNumber);

ucNumber = 255;
usNumber = 65535;
ulNumber = 4294967295;
uiNumber = 4294967295;

//output in diagnostics window
printf("unsigned char:\t%u\r\nunsigned short:\t%u\r\n"
       "unsigned long:\t%u\r\nunsigned int:\t%u\r\n",
       ucNumber,usNumber,ulNumber,uiNumber);

}
```

- 前三行为 *C 动作*的标题。该标题不能更改。
- 在第二部分中，定义变量。为 *char*、*long*、*short* 和 *int* 数据类型及其无符号的对应量各定义一个变量。变量名称前面加上描述数据类型的前缀。这并非必须遵循，但它却使得在程序创建期间能够立即识别变量的数据类型。作为注释，每一行包括变量所需的存储空间(以字符串//开始的注释部分用绿色标记)。
- 在第三部分中，将数值赋给变量。这通过使用赋值运算符=来完成。本实例中所使用的数值恰好是各种数据类型所能显示的数值范围中的极限值。
- 这些数值通过函数 *printf()*在*诊断*窗口中输出。此输出在下一部分中显示。

诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出：

```
Example 1:
char:            -128
short:           -32768
long:            -2147483648
int:             2147483647
unsigned char:   255
unsigned short:  65535
unsigned long:   4294967295
unsigned int:    4294967295
```

4.2.2 实例 2 - 定义的数据类型(整数)

也可以使用专门定义的数据类型来代替 C 中可用的缺省数据类型。然而这些专门定义的数据类型只是实际数据类型的别名。在本实例中，用各种定义的数据类型来显示整数。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 2* 组态了本实例。

Defined Types

按钮 2 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{

CHAR cNumber;           //signed 8 bit value
SHORT sNumber;          //signed 16 bit value
LONG lNumber;           //signed 32 bit value
INT iNumber;            //signed 32 bit value

BOOL bNumber;           //TRUE or FALSE
BYTE byNumber;          //unsigned 8 bit value
WORD wNumber;           //unsigned 16 bit value
DWORD dwNumber;         //unsigned 32 bit value
UINT uiNumber;          //unsigned 32 bit value

cNumber = -128;
sNumber = -32768;
lNumber = -2147483648;
iNumber =  2147483647;

//output in diagnostics window
printf("\r\nExample 2:\r\n");

printf("CHAR:\t\t%d\r\nSHORT:\t\t%d\r\n"
       "LONG:\t\t%d\r\nINT:\t\t%d\r\n",
       cNumber,sNumber,lNumber,iNumber);

bNumber  = TRUE;
byNumber = 255;
wNumber  = 65535;
dwNumber = 4294967295;
uiNumber = 4294967295;

//output in diagnostics window
printf("BOOL:\t\t%u\r\nBYTE:\t\t%u\r\nWORD:\t\t%u\r\n"
       "DWORD:\t\t%u\r\nUINT:\t\t%u\r\n",
       bNumber,byNumber,wNumber,dwNumber,uiNumber);

}
```

- 在第一部分中，定义变量。为 *CHAR*、*SHORT*、*LONG* 和 *INT* 这些已定义的数据类型及其无符号的对应量 *BYTE*、*WORD*、*DWORD* 和 *UINT* 各定义一个变量。此外，定义一个 *BOOL* 数据类型的变量。可以将定义的值 *TRUE* 或 *FALSE* 赋给 *BOOL* 数据类型的变量。与先前的实例一样，这些变量名称的前面加上描述数据类型的前缀。
- 在第二部分中，将数值赋给变量。本实例中所使用的数值又恰好是各种数据类型所能显示的数值范围中的极限值。
- 这些数值通过函数 *printf()*在*诊断*窗口中输出。此输出在下一部分中显示。

诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出：

```
Example 2:
CHAR:       -128
SHORT:      -32768
LONG:       -2147483648
INT:        2147483647
BOOL:       1
BYTE:       255
WORD:       65535
DWORD:      4294967295
UINT:       4294967295
```

类型定义

本节中所使用的数据类型已经用 *typedef* 命令进行定义。以下程序代码说明 *BYTE* 数据类型是如何定义的。*BYTE* 只是 C 中可用的缺省数据类型 unsigned char 的别名。用户也可以定义自己的别名。

```
typedef unsigned char BYTE;
```

下表包含与 C 中可用缺省数据类型相关的已定义数据类型。

定义的数据类型	C 的数据类型
BOOL	int
CHAR	char
SHORT	short
LONG	long
INT	int
BYTE	unsigned char
WORD	unsigned short
DWORD	unsigned long
UINT	unsigned int

4.2.3 实例 3 - WinCC 变量(整数)

在大多数情况下，要通过 *C 动作*或其它函数来使对象动态化和解决类似的事情时，必须使用 WinCC 变量。为此，有许多用于读取和写入 WinCC 变量值的函数可以使用。这些函数可以与每种 WinCC 缺省变量类型一起使用。在本实例中，将数值写入各种 WinCC 变量。WinCC 变量的内容显示在输出域内。在*事件 → 鼠标 → 鼠标动作*处为如下所示的对象*按钮 3* 组态了本实例。



按钮 3 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

CHAR cNumber;            //signed 8 bit value
SHORT sNumber;           //signed 16 bit value
LONG lNumber;            //signed 32 bit value

BOOL bNumber;            //TRUE or FALSE
BYTE byNumber;           //unsigned 8 bit value
WORD wNumber;            //unsigned 16 bit value
DWORD dwNumber;          //unsigned 32 bit value

cNumber = -128;
sNumber = -32768;
lNumber = -2147483648;

//set wincc tags
SetTagSByte("S08i_course_tag_1",cNumber);
SetTagSWord("S16i_course_tag_1",sNumber);
SetTagSDWord("S32i_course_tag_1",lNumber);

bNumber  = TRUE;
byNumber = 255;
wNumber  = 65535;
dwNumber = 4294967295;

//set wincc tags
SetTagBit("BINi_course_tag_1",(SHORT)bNumber);
SetTagByte("U08i_course_tag_1",byNumber);
SetTagWord("U16i_course_tag_1",wNumber);
SetTagDWord("U32i_course_tag_1",dwNumber);

}
```

- 在第一部分中，定义变量。根据 WinCC 变量可用的数据类型选择变量的数据类型。
- 在第二部分中，将数值赋给变量。本实例中所使用的数值又恰好是各种数据类型所能显示的数值范围中的极限值。
- 利用相应的函数将变量值赋给各种 WinCC 变量。函数名称包括文本 *SetTag* 和函数所应用的 WinCC 变量的数据类型标志。与用于写入 WinCC 变量的 *SetTag* 函数相对应，也有用于读取 WinCC 变量的 *GetTag* 函数。
- 如果将 *BOOL* 数据类型(*int* 的别名)的变量传送给 *SetTagBit()*函数，则编译程序将发出警告。发生这种情况是因为 *SetTagBit()*函数希望用 *SHORT* 作为所传送变量的数据类型。因此，在本实例代码中将变量 bNumber 的内容传送给 *SetTagBit()*函数之前，先将其转换为 SHORT 类型。此过程又称为 *Typecast* (类型转换)。

类型转换

变量的内容在传送给函数或赋给其它变量之前，可以转换为不同的数据类型。然而，变量本身的数据类型保持不变。以下程序代码说明了如何将 *float* 数据类型的变量转换为 *int* 数据类型。

```
iNumber = (int)fNumber;
```

WinCC 变量的数据类型

下表包含与 C 中可用数据类型相对应的 WinCC 变量的各种数据类型。它们就是传送给 *SetTag* 函数并由 *GetTag* 函数返回的数据类型。

WinCC 变量的数据类型	C 变量的数据类型
有符号的 8 位数	char
有符号的 16 位数	short int
有符号的 32 位数	long int
二进制变量	short int
无符号的 8 位数	BYTE
无符号的 16 位数	WORD
无符号的 32 位数	DWORD

4.2.4 实例 4 - C 的数据类型(浮点数)

在本实例中，用 C 中可用的缺省数据类型来显示浮点数。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 4*组态了本实例。

C Data Types

按钮 4 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

float fNumber;     //32 bit
double dNumber;    //64 bit

fNumber = 1.0000001;
dNumber = 1.0000001;

//output in diagnostics window
printf("\r\nExample 4:\r\n");

printf("float:\t%2.17f\tsizeof(float):\t%d\r\n"
       "double:\t%2.17f\tsizeof(double):\t%d\r\n",
       fNumber,sizeof(float),dNumber,sizeof(double));

}
```

- 在第一部分中，定义变量。为 *float* 和 *double* 数据类型各定义一个变量。
- 在第二部分中，将数值赋给变量。在本实例中，将相同的数值赋给两个变量。
- *float* 型变量的精度大约为小数点后第七位。*double* 型变量可以显示的精度为浮点型的两倍。这可以参见*诊断窗口*中输出的数值(使用 *printf()*函数)。除变量值之外，还输出其所需的存储空间。变量所需的存储空间通过 *sizeof()*命令来确定。所需的存储空间以字节为单位表示。

诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出：

```
Example 4:
float:    1.00000011920928960    sizeof(float):    4
double:   1.00000010000000010    sizeof(double):   8
```

4.2.5 实例 5 - WinCC 变量(浮点数)

除整数以外，WinCC 变量也可以包含浮点数。因此，与 C 的数据类型 float 和 double 相对应，WinCC 变量有两种数据类型可用。为了以读或写的方式访问这些 WinCC 变量，提供了相应的 *SetTag* 和 *GetTag* 函数。在本实例中，将数值写入各种 WinCC 变量。WinCC 变量的内容显示在输出域内。在*事件 → 鼠标 → 鼠标动作*处为如下所示的对象*按钮 5* 组态了本实例。

WinCC Tags

按钮 5 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

float fNumber;     //32 bit
double dNumber;    //64 bit

fNumber = 1.0000001;
dNumber = 1.0000001;

//set wincc tags
SetTagFloat("F32i_course_tag_1",fNumber);
SetTagDouble("F64i_course_tag_1",dNumber);

}
```

- 在第一部分中，定义变量。为 *float* 和 *double* 数据类型各定义一个变量。
- 在第二部分中，将数值赋给变量。在本实例中，再将相同的数值赋给两个变量。
- 利用相应的函数将变量值赋给各种 WinCC 变量。与此处所用的用于写入 WinCC 变量的 *SetTag* 函数相对应，用于读取 WinCC 变量的 *GetTag* 函数也可用。

4.2.6 实例 6 - 静态变量和外部变量

在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 6*组态了本实例。

Count Mouse Clicks

静态变量

C 变量在定义后才能在函数中生效。在函数终止后，它又变成无效。如果再次调用该函数，则将会再生成 C 变量。然而，如果在变量前面加关键字 *static*，则在两次函数调用之间将会保留该变量。因此，它将保留其值。然而对于 *C 动作*，只有选择了 WinCC 画面，静态变量才会有效。如果撤消选定画面，则静态变量变成无效。再次打开画面后，在首次执行 *C 动作*期间将会再次生成静态变量。

外部变量

C 变量只能在定义它的函数内访问。然而，如果在任何函数之外定义变量，则该变量将成为全局(外部)变量。于是，在任何函数中都可以利用关键字 *extern* 来声明该变量并且可以访问它。

项目函数 CreateExternalTags()

```
int ext_iNumber = 0;

void CreateExternalTags()
{

//nothing to do

}
```

- 函数 *CreateExternalTags()*只用于定义和初始化一个 *int* 类型的外部变量。在项目启动时，调用一次该函数(在起始画面 *cc_0_startpicture_00.PDL* 的*事件* → *其它* → *打开画面处*)。从此刻起，变量 *ext_iNumber* 被定义并且可以在任何 *C 动作*和其它函数中使用。

按钮 6 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert

{
//declare external tag
extern int ext_iNumber;

//define static tag
static int stat_iNumber = 0;

//output in diagnostics window
printf("\r\nExample 6:\r\n"
       "mouseclicks since project was started: %d\r\n"
       "mouseclicks since picture was opend: %d\r\n",
       ++ext_iNumber,++stat_iNumber);

}
```

- 在第一部分中，声明外部变量 *ext_iNumber*，以便能在 *C 动作*中使用它。
- 在第二部分中，定义并初始化静态变量 *stat_iNumber*。它们将在选择 WinCC 画面后首次执行 *C 动作*时执行。对于以后再次执行 *C 动作*，该变量的值将会保留。如果撤消选定后再选择画面，则将会再生成变量。
- 变量的数值通过自增运算符++增加 1，并通过 printf()函数在诊断窗口中输出。因此，变量 ext_iNumber 将显示从项目启动后单击按钮的次数，而变量 stat_iNumber 将显示从画面打开后单击的次数。此输出在下一部分中显示。

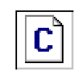
诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出：

```
Example 6:
mouseclicks since project was started: 1
mouseclicks since picture was opend: 1
```

4.3 C 中的运算符和数学函数

在 WinCC 项目 Project_C_Course 中，有关运算符主题的实例可以通过单击如下所示的浏览栏图标来访问。实例在 cc_9_example_01.PDL 画面中组态。



运算符

在程序中，运算符控制变量和常量进行的运算。变量和常量与运算符连接，这样会导致产生新的变量值。
运算符可以分成多种类别。包括数学运算符、按位运算符和赋值运算符。

数学运算符

运算符	描述
+ (单目)	正号(实际可以不用)
- (单目)	负号
+ (双目)	加
- (双目)	减
*	乘
/	除
%	模(返回除法运算的余数)
++	自增
--	自减

按位运算符

这些运算符使得可以对变量中的各个位进行设置、查询或重新设定。

<table>
<tr><th>运算符</th><th>描述</th></tr>
<tr><td>&</td><td>按位与</td></tr>
<tr><td>|</td><td>按位或</td></tr>
<tr><td>^</td><td>按位异或</td></tr>
<tr><td>~</td><td>按位取反</td></tr>
<tr><td><<</td><td>将位向左移</td></tr>
<tr><td>>></td><td>将位向右移</td></tr>
</table>

逻辑运算符

所有的逻辑运算符都遵循相同的原则：0 表示假，所有其它数字都表示真。这些运算符不是生成 0 (假)就是生成 1 (真)。

<table>
<tr><th>运算符</th><th>描述</th></tr>
<tr><td>></td><td>大于</td></tr>
<tr><td>>=</td><td>大于或等于</td></tr>
<tr><td>==</td><td>等于</td></tr>
<tr><td>!=</td><td>不等于</td></tr>
<tr><td><=</td><td>小于或等于</td></tr>
<tr><td><</td><td>小于</td></tr>
<tr><td>&&</td><td>逻辑与</td></tr>
<tr><td>||</td><td>逻辑或</td></tr>
<tr><td>!</td><td>逻辑非</td></tr>
</table>

4.3.1 实例 1 - 基本的数学运算

在本实例中使用了基本的数学运算符。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 1* 组态了本实例。

Arithmetic Operations

按钮 1 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

float fValue1 = 123.6;
float fValue2 = 23.4;

float fResAdd;
float fResSub;
float fResMul;
float fResDiv;

fResAdd = fValue1 + fValue2; //add
fResSub = fValue1 - fValue2; //subtract
fResMul = fValue1 * fValue2; //multiply
fResDiv = fValue1 / fValue2; //divide

//output in diagnostics window
printf("\r\nExample 1\r\n");
printf("%.1f + %.1f = %.1f\r\n",fValue1,fValue2,fResAdd);
printf("%.1f - %.1f = %.1f\r\n",fValue1,fValue2,fResSub);
printf("%.1f * %.1f = %.1f\r\n",fValue1,fValue2,fResMul);
printf("%.1f / %.1f = %.1f\r\n",fValue1,fValue2,fResDiv);

}
```

- 在第一部分中，定义并初始化两个数据类型为 *float* 的变量。将数学运算符应用于这两个变量。
- 在第二部分中，另外定义四个数据类型为 *float* 的变量。这些变量存储执行数学运算后的结果。
- 在第三部分中，用数学运算符进行加、减、乘和除运算。
- 这些计算结果通过 *printf()*函数在*诊断窗口*中输出。此输出在下一部分中显示。

诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出：

```
Example 1
123.6 + 23.4 = 147.0
123.6 - 23.4 = 100.2
123.6 * 23.4 = 2892.2
123.6 / 23.4 = 5.3
```

4.3.2 实例 2 – 自增和自减运算符

在本实例中使用了自增和自减运算符。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 2*组态了本实例。

Operator ++ and --

前缀和后缀

自增和自减运算符既可以用作前缀也可以用作后缀。这两种类型执行相同的动作，也就是使应用运算符的变量值增加或减少 1。其区别在于返回值。如果运算符作为前缀，则增加或减少变量值并返回此新值。如果运算符作为后缀，则返回原来的变量值，然后才使变量递增或递减。

```
iValue = ++iCount; //präfix
iValue = iCount++; //postfix
```

按钮 2 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

static int stat_iPräfix = 0;
static int stat_iPostfix = 0;

printf("\r\nExample 2\r\n");

//execute operators
printf("Prefix operator on first tag:  %d\r\n",++stat_iPräfix);
printf("Postfix operator on second tag: %d\r\n",stat_iPostfix++);

//check values
printf("Value of first tag after execution: %d\r\n",stat_iPräfix);
printf("Value of second tag after execution: %d\r\n",stat_iPostfix);

}
```

- 在第一部分中，定义并初始化两个数据类型为 int 的变量。
- 自增运算符作为前缀或后缀应用于这两个变量。这些运算符的返回值通过 *printf()*函数在诊断窗口中输出。然后变量内容也通过 *printf()*函数在*诊断窗口*中输出。此输出在下一部分中显示。

诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出：可以看到自增运算符作为前缀则返回增加后的变量值，而作为后缀则返回原来的变量值。但是，这两种情况都会使变量递增。

```
Example 2
Prefix operator on first tag:  1
Postfix operator on second tag: 0
Value of first tag after execution: 1
Value of second tag after execution: 1
```

4.3.3 实例 3 - 位运算

在本实例中使用了基本的按位运算符。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 3* 组态了本实例。

Bit Operations

说明

在本实例中，按位运算符应用于两个 WinCC 变量(*无符号的 16 位数*)的内容。运算的结果存储在另一个相同类型的 WinCC 变量中。应用的运算符由对象*按钮 6* 控制并同时显示。按位连接 *AND*、*OR*、*NAND*、*NOR* 和 *EXOR* 都可用。为每个选项分配一个数值，并将它存储在另一个 WinCC 变量(*无符号的 8 位数*)中。

按钮 3 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{

BYTE byOperation;

DWORD dwValue1;
DWORD dwValue2;
DWORD dwResult;

//read tag values
dwValue1 = GetTagWord("U16i_course_op_1");
dwValue2 = GetTagWord("U16i_course_op_2");

//get desired operation
byOperation = GetTagByte("U08i_course_op_1");

switch (byOperation)
{
	//AND
	case 0: dwResult = dwValue1 & dwValue2;
		break;
	//OR
	case 1: dwResult = dwValue1 | dwValue2;
		break;
	//NAND
	case 2: dwResult = ~(dwValue1 & dwValue2);
		break;
	//NOR
	case 3: dwResult = ~(dwValue1 | dwValue2);
		break;
	//EXOR
	case 4: dwResult = dwValue1 ^ dwValue2;
		break;
	//???
	default:return;
}

//write result
SetTagWord("U16i_course_op_3",(WORD)dwResult);

}
```

- 在第一部分中，定义一个数据类型为 *BYTE* 的变量以及三个数据类型为 *DWORD* 的变量。这些变量用于临时存储 WinCC 变量。
- 在第二部分中，把要连接的两个 WinCC 变量读入变量 *dwValue1* 和 *dwValue2* 中。另外，确定按位连接运算符类型的 WinCC 变量将被读入变量 *byOperation* 中。
- 在第三部分中，根据变量 *byOperation* 的内容按位连接变量 *dwValue1* 和 *dwValue2*。连接结果存储在变量 *dwResult* 中。要执行的连接运算通过 *switch-case* 结构来选择。该结构在*循环*一章中进行了详细描述。
- 在第四部分中，变量 *dwResult* 包含的连接结果写入相应的 WinCC 变量中。

4.3.4 实例 4 - 按字节循环移动

在本实例中，按位移动运算符用于使 WinCC 变量(*无符号的 16 位数*)中包含的数值按字节循环移动。也就是说交换高位字节和低位字节。在*事件 → 鼠标 → 鼠标动作*处为如下所示的对象*按钮 4* 组态了本实例。

Rotate

按钮 4 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

DWORD dwValue;

DWORD dwtempValue1;
DWORD dwtempValue2;

//read tag value
dwValue = GetTagWord("U16i_course_op_3");

//rotate bytes
dwtempValue1 = dwValue<<8;
dwtempValue2 = dwValue>>8;

dwValue = dwtempValue1 | dwtempValue2;

//write result
SetTagWord("U16i_course_op_3",(WORD)dwValue);

}
```

- 在第一部分中，定义一个数据类型为 *DWORD* 的变量。此变量用于临时存储 WinCC 变量。此外，再定义两个类型为 *DWORD* 的辅助变量。
- 在第二部分中，把要进行处理的 WinCC 变量写入变量 *dwValue* 中。
- 在第三部分中，变量 *dwValue* 的各个位向左移动 8 位(一个字节)，然后存储在变量 *dwtempValue1* 中。接着将变量 *dwValue* 的各个位向右移动 8 位，然后存储在变量 *dwtempValue2* 中。此处确定的两个数值按位连接(OR)，并且将结果存储在变量 *dwValue* 中。
- 在第四部分中，将变量 *dwValue* 所包含的循环移动后的变量值写入相应的 WinCC 变量中。

4.3.5 实例 5 - 数学函数

在本实例中使用了缺省情况下在 C 语言中可使用的各种数学函数。在*事件* → *鼠标* → *鼠标动作处*为如下所示的对象*按钮 5* 组态了本实例。

Calculate

按钮 5 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

double dValue = 123.6;
int iValue = -24;

double dResPow;
double dResSqrt;
int iResAbs;
int iResRand;

dResPow = pow(dValue,3); //power of 3
dResSqrt= sqrt(dValue);  //square root
iResAbs = abs(iValue);   //absolute
iResRand= rand();        //random

//output in diagnostics window
printf("\r\nExample 5\r\n");
printf("%.1f raised to the power of 3 = %.1f\r\n",dValue,dResPow);
printf("Square root of %.1f\t = %.1f\r\n",dValue,dResSqrt);
printf("Absolute value of %d\t = %d\r\n",iValue,iResAbs);
printf("A pseudorandom number\t = %d\r\n",iResRand);

}
```

- 在第一部分中，定义变量。
- 首先调用 *pow()*函数。为该函数分配两个参数。在本实例中，函数的返回值等于 *dValue* 变量值的三次方。
- 接着调用 *sqrt()*函数。此函数的返回值等于传送值的平方根。
- 再接着调用 *abs()*函数。此函数的返回值等于传送值的绝对值。
- 然后再调用 *rand()*函数。没有参数分配给此函数。它将返回一个随机值作为返回值。
- 这些计算结果通过 *printf()*函数在诊断窗口中输出。此输出在下一部分中显示。

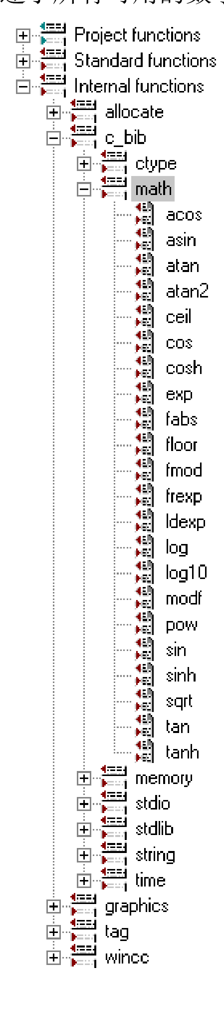
诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出：

```
Example 5
123.6 raised to the power of 3 = 1888232.3
Square root of 123.6           = 11.1
Absolute value of -24          = 24
A pseudorandom number          = 41
```

其它数学函数

在函数选择中，数学函数可以在*内部函数* → *c_bib* → *math* 下找到。下图描述了所有可用的数学函数(用灰色阴影标记)。



4.4 指针

在 WinCC 项目 Project_C_Course 中，有关指针主题的实例可以通过单击如下所示的浏览栏图标来访问。实例在 cc_9_example_02.PDL 画面中组态。



使用指针

指针是 C 语言的重要组件。指针是包含地址的变量，通常该地址是另一个变量的地址。
定义指针就象定义普通变量一样。但是指针指向的数据类型名称要添加单目字符*。不得将此字符误认为是用于乘法运算的双目运算符*。在以下程序代码中，定义了 int 数据类型的指针变量。

```
int* piValue;
```

指针的内容没有定义。它仍然指向一个无效的 int 数据类型的变量。为了澄清这一点，在定义指针时应该用数值 NULL 进行初始化。在应用指针前可检查其有效性。

```
int* piValue = NULL;
```

要使指针指向 int 数据类型的变量，必须将变量的地址分配给它。这通过单目运算符来完成。单目运算符又称为地址运算符。此运算符返回变量地址，而不是变量值。在以下程序代码中，将数据类型为 int 的变量的地址分配给指针。

```
piValue = &iValue;
```

可以通过单目运算符* (也称为内容运算符)来实现对指针所指向的变量值的访问。在以下程序代码中，将指针所指向的变量值分配给一个数据类型为 int 的变量。

```
iValue = *piValue;
```

使用向量

指针和向量密切相关。在以下程序代码中，定义了一个由 5 个 int 数据类型的变量组成的向量。

```
int iVector[5];
```

向量的各个元素可以通过其下标来访问。在以下程序代码中，访问最后一个向量元素的内容。这通过下标运算符[ ]来完成。

```
iValue = iVector[4];
```

向量名也可用作指向第一个向量元素的指针。可通过将此指针移动几个元素来访问某个向量元素。如以下程序代码所示，它通过将指针加上一个 int 数值来完成。结果指针的内容通过内容运算符*来访问。同前所示，访问的是最后一个向量值。

```
iValue = *(iVector+4);
```

字符串

在 C 语言中，字符串可以定义为由字符组成的向量或指向字符的指针。除代码字符之外，C 语言还在字符串的结尾添加一个空字符。它作为字符串的结束符。在如下所示的程序代码中，定义了两种类型的字符串变量。

```
char* lpszString   = "String1";
char  szString[10] = "String2";
```

以下所示为两种字符串变量的内部显示。在第一种情况中，为字符串变量保留的内存空间恰好与初始化所指示的字符串所需的空间一样大。在第二种情况中，保留的内存空间与定义向量时指定的空间一样大。

S	t	r	i	n	g	1	\0

S	t	r	i	n	g	2	\0	?	?

4.4.1 实例 1 - 指针

在本实例中执行基本的指针运算。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 1* 组态了本实例。

Pointers

按钮 1 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

int iValue1 = 126;
int iValue2 =  23;

//declare and initialize pointer
int* piValue = NULL;

printf("\r\nExample 1\r\n");
printf("Address: %x\tValue: undefined\r\n",piValue);

//point at iValue1
piValue = &iValue1;
printf("Address: %x\tValue: %d\r\n",piValue,*piValue);

//point at iValue1
piValue = &iValue2;
printf("Address: %x\tValue: %d\r\n",piValue,*piValue);

}
```

- 在第一部分中，定义并初始化两个数据类型为 *int* 的变量。
- 在第二部分中，定义一个指向 *int* 数据类型的变量的指针，并用 *NULL* 对其进行初始化。
- 在第三部分中，通过 *printf()*函数输出该指针中包含的地址。当前该指针指向的内容没有定义。此时通过内容运算符*访问指针的内容会引起一般的访问违例。
- 在第四部分中，将变量 *iValue1* 的地址赋给指针。通过 *printf()*函数再次输出其地址和内容。
- 在第五部分中，将变量 *iValue2* 的地址赋给指针，并且再次输出结果。此程序的输出在下一部分显示。

诊断窗口中的输出

本节中描述的实例在诊断窗口内生成下列输出：

```
Example 1
Address: 0          Value: undefined
Address: 2b4f9f8    Value: 126
Address: 2b4f9fc    Value: 23
```

4.4.2 实例 2 - 向量

在本实例中执行基本的向量运算。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象按钮 2 组态了本实例。

Vectors

按钮 2 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{

//declare and initialize int vector
int iValue[5] = { 10, 20, 30, 40, 50 };

int iIndex;

printf("\r\nExample 2\r\n");

//access vector elements
for (iIndex = 0; iIndex<5; iIndex++)
{
      printf("Index: %d\t Value: %d\r\n",iIndex,iValue[iIndex]);
}

}
```

- 在第一部分中，定义一个由 5 个 *int* 数据类型的变量组成的向量。向量在定义时就已用数值进行初始化。
- 在第二部分中，定义 *int* 数据类型的计数器变量 *iIndex*。
- 向量的各个元素通过 *printf()*函数输出。对各个元素的访问在一个 *for* 循环中通过下标运算符[ ]来完成。涉及循环的内容在下一章*循环*中描述。此输出在下一部分中显示。

诊断窗口中的输出

本节中描述的实例在诊断窗口内生成下列输出结果：

```
Example 2
Index: 0    Value: 10
Index: 1    Value: 20
Index: 2    Value: 30
Index: 3    Value: 40
Index: 4    Value: 50
```

4.4.3 实例 3 - 指针与向量

在本实例中解释指针与向量之间的关系。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 3* 组态了本实例。

Pointers and Vectors

按钮 3 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

int iValue[] = { 10, 20, 30, 40, 50 };

int iIndex;

int* piElement = NULL;

printf("\r\nExample 3\r\n");

/////////////////////////////////////////////////////////////////////
//access via seperat pointer

//point to the first element
piElement = &iValue[0];

printf("Startaddress: %x\r\n",piElement);

for (iIndex = 0; iIndex<5; iIndex++)
{
      printf("Index: %d\t Value: %d\r\n",iIndex,*(piElement+iIndex));
}

printf("\r\n");

/////////////////////////////////////////////////////////////////////
//access without seperate pointer
printf("Startaddress: %x\r\n",iValue);

for (iIndex = 0; iIndex<5; iIndex++)
{
      printf("Index: %d\t Value: %d\r\n",iIndex,*(iValue+iIndex));
}

}
```

- 在第一部分中，定义一个由 5 个 *int* 数据类型的变量组成的向量。向量在定义时就已用数值进行初始化。在这种情况下，定义向量时可以省略大小规定。
- 在第二部分中，定义 *int* 数据类型的计数器变量 *iIndex*。
- 在第三部分中，为 *int* 数据类型的变量定义指针 *piElement*，并用 *NULL* 进行初始化。
- 在第四部分中，将第一个向量元素的地址赋给指针 *piElement*。此地址通过 *printf()* 函数输出。

- 在第五部分中，通过指针 *piElement* 访问向量的各个元素。在一个 *for* 循环中通过将指针指向各个元素，并通过内容运算符*来进行访问。
- 在第六部分中，再次访问向量的各个元素。但是这次将向量名本身用作指针。此程序的输出在下一部分中显示。

诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出结果：

```
Example 3
Startaddress: 2b4f9e8
Index: 0    Value: 10
Index: 1    Value: 20
Index: 2    Value: 30
Index: 3    Value: 40
Index: 4    Value: 50

Startaddress: 2b4f9e8
Index: 0    Value: 10
Index: 1    Value: 20
Index: 2    Value: 30
Index: 3    Value: 40
```

4.4.4 实例 4 - 字符串

在本实例中解释对字符串变量的使用。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 4* 组态了本实例。



按钮 4 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

//declare and initialize string
char szText[13] = "example text";

int i;

printf("\r\nExample 4\r\nCharacters:\r\n");

//access single characters
for (i=0; i<12; i++)
{
        printf("[%c],",szText[i]);
}

printf("\r\n");

//access hole string
printf("String:\r\n%s\r\n",szText);

}
```

- 在第一部分中，定义一个字符串(由 13 个字符组成的向量)。此字符串的长度比分配的初始化字符串多一个字符，以便为结束空字符留出空间。
- 在第二部分中，定义 int 数据类型的计数器变量 i。
- 在第三部分中，通过 *printf()*函数输出字符串的各个字符。在 *for* 循环中通过下标运算符[ ]来对这些字符进行访问。
- 在第四部分中，通过 *printf()*函数输出整个字符串。此程序的输出在下一部分中显示。

诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出：

```
Example 4
Characters:
[e],[x],[a],[m],[p],[l],[e],[ ],[t],[e],[x],[t],
String:
example text
```

4.4.5 实例 5 - WinCC 文本变量

在本实例中解释了 C 中的字符串变量与 WinCC 文本变量之间的关系。在*事件 → 鼠标 → 鼠标动作*处为如下所示的对象*按钮 5*组态了本实例。

WinCC Text Tags

按钮 5 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

//declare and initialize pointer to string
char* pszText = NULL;

//get wincc tag value
pszText = GetTagChar("T08i_course_point_1");

printf("\r\nExample 5\r\n");

//access string
printf("String: %s\r\nStringlength: %d\r\nStartaddress: %x\r\n",
      pszText,strlen(pszText),pszText);

}
```

- 在第一部分中，定义一个字符串(指向第一个字符的指针)。用 *NULL* 初始化此字符串。
- 在第二部分中，通过 *GetTagChar()*函数读入 WinCC 文本变量的内容。当返回字符串的起始地址时，函数保留字符串所需的内存空间。
- 在第三部分中，通过 *printf()*函数输出整个字符串。此外，字符串的长度由 *strlen()*函数来确定，并且与字符串的起始地址一起输出。此程序的输出在下一部分中显示。

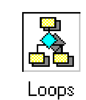
诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出：

Example 5
String: This is an example text!
Stringlength: 24
Startaddress: 1682828

4.5 循环和条件语句

在 WinCC 项目 Project_C_Course 中，有关循环主题的实例可以通过单击如下所示的浏览栏图标来访问。实例在 cc_9_example_03.PDL 画面中组态。



循环

只要条件满足，循环可用于重复执行一个代码段。
通常，有两种循环类型：预检查循环和后检查循环。预检查循环在要执行循环的循环体之前进行检查。后检查循环在要执行循环的循环体之后进行检查。因此，后检查循环至少要执行一次。
循环可以分为下列类型。

while

while 循环的实例显示如下。只要条件满足，就重复执行循环。在本实例中，只要变量 i 的值小于 5 就执行循环。

```
int i = 0;

while (i < 5)
{
	//do something
	++i;
}
```

do - while

do-while 循环的实例显示如下。该循环至少执行一次，然后只要条件满足就重复执行。在本实例中，只要变量 i 的值小于 5 就执行循环。

```
int i = 0;

do
{
	//do something
	++i;
}
while (i < 5);
```

for

for 循环的实例显示如下。只要条件满足，就重复执行循环。循环计数器的初始化以及循环计数器的运算过程可以在循环内用公式表示。

```
int i = 0;

for (i=0, i<5, i++)
{
        //do something
}
```

条件语句

在循环中，只要条件为真，就处理循环体。在条件语句中，如果条件为真，语句只处理一次。
条件语句可以分为下列类型。

if-else

如果条件为真，就处理 if 分支中的语句。如果条件不合适，就执行 else 分支中的语句。如果没有另一个要执行的语句，也可以省略 else 分支。

```
if (i < 5)
{
        //do something
}
else
{
        //do something else
}
```

switch-case

在这种情况下，检查变量是否匹配。Switch 指定要检查的变量。程序检查哪一个 case 分支与变量的值一致。然后执行该 case 分支。可以定义任意个 case 分支。每个 case 分支必须以 break 结束。可以选择插入 default 分支。如果要检查的变量的值与任何 case 分支都不一致，则将执行此分支。

```
switch (i)
{
        case 0: //do something
                break;
        case 1: //do something
                break;
        default://do something default
                break;
}
```

4.5.1 实例 1 - while 循环

在本实例中，解释 while 循环的应用。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 1* 组态了本实例。



按钮 1 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

//loop count
int iCount = 0;

printf("\r\nExample 1\r\n");

//while loop
while (iCount < 5)
{

	//do something
	printf("Executed loop: iCount = %d\r\n",iCount);

	++iCount;

}

printf("Exit loop: iCount = %d\r\n",iCount);


}
```

- 在第一部分中，定义并初始化 *int* 数据类型的计数器变量 *iCount*。
- 接着，编写 while 循环。只要计数器变量 *iCount* 的值小于 5，就执行该循环。每次执行该循环时，通过 *printf()*函数进行输出。在循环结束时，计数器变量 *iCount* 增加 1。此程序的输出在下一部分中显示。

诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出：

```
Example 1
Executed loop: iCount = 0
Executed loop: iCount = 1
Executed loop: iCount = 2
Executed loop: iCount = 3
Executed loop: iCount = 4
Exit loop: iCount = 5
```

4.5.2 实例 2 - do-while 循环

在本实例中，解释 *do-while* 循环的应用。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 2* 组态了本实例。

do - while

按钮 2 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

//loop count
int iCount = 0;

printf("\r\nExample 2\r\n");

//do-while loop
do
{

      //do something
      printf("Executed loop: iCount = %d\r\n",iCount);

      ++iCount;

}
while (iCount < 5);

printf("Exit loop: iCount = %d\r\n",iCount);

}
```

- 在第一部分中，定义并初始化 *int* 数据类型的计数器变量 *iCount*。
- 接着，编写 do-while 循环。只要计数器变量 *iCount* 的值小于 5，就执行该循环。但是，因为此条件只有在执行循环之后才检查，所以该循环至少执行一次。每次执行循环时，通过 *printf()*函数进行输出。在循环结束时，计数器变量 *iCount* 增加 1。此程序的输出在下一部分中显示。

诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出：

Example 2
Executed loop: iCount = 0
Executed loop: iCount = 1
Executed loop: iCount = 2
Executed loop: iCount = 3
Executed loop: iCount = 4
Exit loop: iCount = 5

4.5.3 实例 3 - for 循环

在本实例中，解释 *for* 循环的应用。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 3* 组态了本实例。



按钮 3 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

//loop count
int iCount = 0;

printf("\r\nExample 3\r\n");

//for loop
for (iCount=0; iCount<5; iCount++)
{

	//do something
	printf("Executed loop: iCount = %d\r\n",iCount);

}

printf("Exit loop: iCount = %d\r\n",iCount);

}
```

- 在第一部分中，定义并初始化 *int* 数据类型的计数器变量 *iCount*。
- 接着，编写 *for* 循环。只要计数器变量 *iCount* 的值小于 5，就执行该循环。计数器变量的初始化直接编写在循环的调用中，如同使计数器变量递增的动作一样。每次执行循环时，通过 *printf()*函数进行输出。此程序的输出在下一部分中显示。

诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出：

```
Example 3
Executed loop: iCount = 0
Executed loop: iCount = 1
Executed loop: iCount = 2
Executed loop: iCount = 3
Executed loop: iCount = 4
Exit loop: iCount = 5
```

4.5.4 实例 4 - 无限循环

在本实例中，解释无限循环。在大多数情况下，这些循环是由于编程错误而无意识创建的，其循环条件始终保持为真。然而，也可以有意识地应用它们。在这种情况下，必须利用其它方式实现循环的终止，即通过 *break* 语句。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 4* 组态了本实例。

Endless Loops

按钮 4 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{
//max loop executions
#define MAX_COUNT 1000000

//loop count
int iCount = 0;

int iProgressBar = 1;
char szProgressText[5];

//endless loop
//another possible loop while(TRUE) {...}
for (;;)
{
	if (iCount > MAX_COUNT)
	{
		break;
	}

	++iCount;

	if (iCount-(iProgressBar*MAX_COUNT/100) != 0)
	{
		continue;
	}

	//set value of progress bar
	SetWidth(lpszPictureName,"ProgressBar",(int)(iProgressBar*2.7));

	//set progress text
	sprintf(szProgressText,"%d%%",iProgressBar);
	SetText(lpszPictureName,"ProgressText",szProgressText);

	++iProgressBar;
}

}
```

- 在第一部分中，定义符号常量 *MAX_COUNT*。该常量代表以下无限循环最多执行的次数。
- 在第二部分中，定义并初始化 *int* 数据类型的计数器变量 *iCount*。
- 当前循环执行的次数要通过进程显示来表示。该显示由一个棒图组成，其长度包含变量 *iProgressBar* 和静态文本，其内容包含字符串变量 *szProgressText*。
- 接着，编写无限循环。该循环也可以利用 *while (TRUE)*语句公式化。

- 在该循环中，检查计数器变量 *iCount*。如果该变量超过 *MAX_COUNT* 的值，则通过 *break* 语句退出循环。
- 计数器变量 *iCount* 将会递增。
- 进程显示表示循环已经执行的百分比。对于每次达到新的百分比，都再次设置进程显示的数值。如果还没有达到新的百分比，则立即再次通过 *continue* 语句执行循环，并且跳过剩余的行。
- 进程显示的数值通过用 *SetWidth()*函数设置棒图 *ProgressBar* 的宽度以及用 *SetText()*函数设置静态文本 *ProgressText* 的文本来进行设置。使用的文本用 *sprintf()*函数来组态。该函数遵循 *printf()*的原理。然而，该文本不通过*全局脚本诊断*窗口输出，而是写入字符串变量。此字符串变量必须定义为函数的第一个参数。

4.5.5 实例 5 - if-else 语句

在本实例中，解释 if-else 语句的应用。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 5* 组态了本实例。



按钮 5 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

BYTE byValue;

//get value ti check
byValue = GetTagByte("U08i_course_loop_1");

printf("\r\nExample 5\r\n");

if (byValue < 5)
{

	//do something
	printf("byValue < 5\r\n");

}
else
{

	//do something
	printf("byValue >= 5\r\n");

}

}
```

- 在第一部分，定义一个 *BYTE* 数据类型的变量 *byValue*。在该变量中，存储 WinCC 变量的内容。
- 在第二部分，使用 *GetTagByte()*函数将 WinCC 变量的内容读入变量 *byValue* 中。
- 在第三部分，编写 *if-else* 语句。这个语句根据变量 *byValue* 的内容通过 *printf()*函数输出。

4.5.6 实例 6 - switch-case 语句

在本实例中解释 switch-case 语句的应用。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 6*组态了本实例。

switch - case

按钮 6 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

BYTE byValue;

//get value to check
byValue = GetTagByte("U08i_course_loop_1");

printf("\r\nExample 6\r\n");

switch (byValue)
{

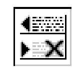
	case 0:		//do something
			printf("byValue = 0\r\n");
			break;
	case 1:		//do something
			printf("byValue = 1\r\n");
			break;
	case 2:
	case 3:
	case 4:		//do something
			printf("byValue = 2,3 or 4\r\n");
			break;
	default:	//do something
			printf("byValue != 0,1,2,3 and 4\r\n");
			break;
}

}
```

- 在第一部分中，定义 *BYTE* 数据类型的变量 *byValue*。在该变量中存储 WinCC 变量的内容。
- 在第二部分中，用 *GetTagByte()*函数将 WinCC 变量的内容读入变量 *byValue* 中。
- 接着，编写 *switch-case* 语句。此语句根据变量 *byValue* 的内容，通过 *printf()*函数进行输出。对于要检查的变量的几个不同数值，如果要执行相同的语句，则相应的 *case* 分支必须相互排列在一起。要执行的语句在最后一个 *case* 分支中编写。

4.6 函数

在 WinCC 项目 Project_C_Course 中，有关函数主题的实例可以通过单击如下所示的浏览栏图标来访问。实例在 cc_9_example_05.PDL 画面中组态。



函数

函数可以使程序代码构造得更好。对于经常重复的语句，不必一次又一次地进行编写，它们可以移入一个函数。这样会形成一个编辑程序代码的中央单元，从而易于维护。
在 WinCC 中，函数可以创建为项目函数或标准函数。

传送参数

可以向函数传送数值，函数根据这些数值将执行相应的语句。这些数值可以用多种不同的方法进行传送。

- 常数可以传送。
- 变量可以传送。只是将变量的值传送给函数。函数不可以访问变量本身。
- 指针可以传送。这使得函数可以访问指针指向的变量。向量和结构只能通过指针分配给函数。

返回值

函数可以只执行语句而不返回数值。如果是这样，则返回值的数据类型为 void。但如果是执行计算，则确定的数值可以通过返回值返回给函数的调用者。如果是这样，则可以返回数值或其它地址。
把数值返回给调用者的另一个选择是将其写入传送的地址区域。向量或结构只能用这种方式来返回。

4.6.1 实例 1 - 数值参数的传送

在本实例中，将创建一个简单函数，用来计算三个数的平均值。参数以数值的形式传送给函数，结果也以数值的形式返回。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 1* 组态了本实例。

Value Parameter

项目函数 MeanValue()

```
double MeanValue(double dValue1, double dValue2, double dValue3)
{
	double dMeanValue;

	dMeanValue = (dValue1+dValue2+dValue3)/3;

	return dMeanValue;

}
```

- 在函数标题内，将函数的名称指定为 *MeanValue()*。将三个 *double* 数据类型的变量传送给函数。返回的也将是一个 *double* 数据类型的变量。
- 接下来，将定义一个 *double* 数据类型的变量，返回的值将存储在该变量中。对所传送的三个值进行累加，然后，将结果除以 3，即得到该返回值。
- 通过 *return* 语句，将结果返回给函数的调用者。

按钮 1 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

double dValue1 = 126.2;
double dValue2 =  23.9;
double dValue3 =  45.7;

double dMeanValue;

//calculate mean value
dMeanValue = MeanValue(dValue1, dValue2, dValue3);

//output into diagnostics window
printf("\r\nExample 1\r\n");

printf("The mean value of %.1f, %.1f and %.1f = %.1f\r\n",
       dValue1,dValue2,dValue3,dMeanValue);

}
```

- 在第一部分中，对 *double* 数据类型的三个变量进行了定义，并进行了初始化。同时，计算这三个变量的平均值。定义另一个 *double* 数据类型的变量来存放该计算结果。
- 使用先前创建的函数 *MeanValue()*来计算所传送变量的平均值。
- 通过 *printf()*函数将此计算结果输出。此输出将在下一节中显示。

诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出结果：

```
Example 1
The mean value of 126.2, 23.9 and 45.7 = 65.3
```

4.6.2 实例 2 - 地址参数的传送

在本实例中，将创建一个简单函数，用于计算任意长度向量的元素的平均值。向量的地址及其长度均传送给函数。计算结果作为数值返回。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 2* 组态了本实例。

Address Parameter

项目函数 MeanValueVector()

```
double MeanValueVector(double* dValue, DWORD dwSize)
{
	double dSum = 0.0;
	int i;

	for(i=0; i<dwSize; i++)
	{
		dSum = dSum + dValue[i];
	}

	return (dSum / dwSize);

}
```

- 在函数标题内，将函数的名称指定为 *MeanValueVector()*。指向 *double* 数据类型变量的指针传送给该函数。该指针指向所期望的向量的第一个元素。另外，向量的长度也传送给该函数。该函数返回一个 *double* 数据类型的变量。
- 接下来，将定义一个 *double* 数据类型的变量，并对其进行初始化。在此变量中，存放所传送向量的元素的总和。使用一个 *for* 循环来计算该总和。
- 通过 *return* 语句，将计算结果返回给该函数的调用者。此计算结果与向量元素的总和除以向量元素的数目相对应。

按钮 2 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

//define  and initialize double vector
double dValue[3] = { 126.2, 23.9, 45.7 };

double dMeanValue;

//calculate mean value of vector
dMeanValue = MeanValueVector(dValue, 3);

//output into diagnostics window
printf("\r\nExample 2\r\n");

printf("The mean value of %.1f, %.1f and %.1f = %.1f\r\n",
       dValue[0],dValue[1],dValue[2],dMeanValue);

}
```

- 在第一部分中，定义了由三个 *double* 数据类型的变量所组成的向量，并对其进行了初始化。同时，将计算这三个变量的平均值。另外还定义了一个 *double* 数据类型的变量，用来存放计算的结果。
- 使用先前所创建的函数 *MeanValueVector()*，可计算出所传送向量元素的平均值。
- 通过 *printf()*函数，可输出计算的结果。此输出函数将在下一节中显示。

诊断窗口中的输出

本节中所描述的实例在*诊断窗口*内生成下列输出：

Example 2
The mean value of 126.2, 23.9 and 45.7 = 65.3

4.6.3 所传送地址域的写入

在本实例中，将创建一个简单函数，该函数用随机数来填充任意长度的向量。向量的地址及其长度均传送给函数。如果通过一个 *BOOL* 类型的变量能够成功地执行该动作，则该函数将以返回值的形式显示。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 3* 组态了本实例。

Writing to the
Address Area

项目函数 FillVector()

```
BOOL FillVector(int* piVector, DWORD dwSize)
{

int i;

//check received pointer
if (piVector == NULL)
{
	return FALSE;
}

//fill vector
for (i=0; i<dwSize; i++)
{
	piVector[i] = rand();
}

return TRUE;

}
```

- 在函数标题内，将函数的名称指定为 *FillVector()*。指向 int 数据类型变量的指针将被传送给该函数。该指针指向所期望的向量的第一个元素。另外，向量的长度也将传送给该函数。返回一个 *BOOL* 数据类型的变量，用来说明是否成功地执行了该函数。
- 接下来，将定义一个 *int* 数据类型的计数器变量。
- 接着，对所传送的指针进行检查。函数调用者则负责传送正确的向量长度。如果传送了不正确的值，将可能导致一个常规的访问冲突。
- 使用 *for* 循环，*rand()*函数用随机数对被传送的向量的元素进行填充。

按钮 3 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

#define VECTOR_SIZE 5

//define  int vector
int iVector[VECTOR_SIZE];

int i;

printf("\r\nExample 3\r\n");

//fill vector
if (FillVector(iVector,VECTOR_SIZE) == FALSE)
{
      printf("Error in FillVector\r\n");
      return;
}

printf("Vector Elements: ");

for (i=0; i<VECTOR_SIZE; i++)
{
      printf("[%d] ",iVector[i]);
}

printf("\r\n");

}
```

- 在第一部分中，定义了一个用于向量元素数目的符号常量 *VECTOR_SIZE*。
- 接下来，将定义一个向量 *iVector*，它由 *int* 数据类型的 *VECTOR_SIZE* 变量所组成。
- 接着，还将定义一个 *i* 计数器变量，该变量为 *int* 数据类型。
- 使用先前所创建的 *FillVector()*函数，可用随机数来填充所传送的 *iVector* 向量的元素。在调用 *FillVector()*函数时对其返回值进行校验可借助于 *if* 语句。
- *iVector* 向量的各个元素可通过 *printf()*函数来输出。该输出函数将在下一节中显示。

诊断窗口中的输出

本节中所描述的实例在诊断窗口内生成下列输出：

```
Example 3
Vector Elements: [18467] [6334] [26500] [19169] [15724]
```

4.6.4 结果地址的返回

在本实例中，将创建一个简单函数，该函数将使用随机数来填充向量。所期望向量的长度将作为一个参数传送给该函数。作为返回值，函数将提供所创建向量的第一个元素的地址。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 4* 组态了本实例。

项目函数 GetFilledVector()

```
int* GetFilledVector(DWORD dwSize)
{

int* piVector = NULL;

int i;

//allocate memory for vector
piVector = SysMalloc(sizeof(int) * dwSize);

//check return value of SysMalloc()
if (piVector == NULL)
{
	return NULL;
}

//fill vector
for (i=0; i<dwSize; i++)
{
	piVector[i] = rand();
}

return piVector;

}
```

- 在函数标题内，将函数的名称指定为 *GetFilledVector()*。将所创建的向量的元素数目传送给该函数。指向 *int** 数据类型的第一个向量元素的指针将返回。
- 接下来，将定义一个 *piVector* 指针，该指针用于 *int* 数据类型的变量，并可使用 *NULL* 对其初始化。
- 接着，将定义一个 i 计数器变量，该变量为 *int* 数据类型。
- 必须为该向量保留足够的存储空间。这可由内部函数 *SysMalloc()*来保证。对于该函数，所传送的期望的存储块大小可通过 *int* 数据类型的变量所需要的存储空间乘以所期望的向量元素的数目来计算。如果可用的存储空间不够，则该函数将返回所保留的存储块的地址或 *NULL*。
- 接下来，将对 *SysMalloc()*函数发出的地址进行校验。如果没有足够的存储空间可供使用，则函数终止，并返回 *NULL*。
- 使用 *for* 循环，可利用 *rand()*函数，用随机数对向量的元素进行填充。
- 通过 *return* 语句，将所创建的向量的地址返回给函数的调用者。

按钮 4 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{

#define VECTOR_SIZE 5

//declare pointer to save address of int vector
int* piVector = NULL;

int i;

printf("\r\nExample 4\r\n");

//get address of filled vector
piVector = GetFilledVector(VECTOR_SIZE);

if (piVector == NULL)
{
	printf("Error in GetFilledVector\r\n");
	return;
}

printf("Vector Elements: ");

for (i=0; i<VECTOR_SIZE; i++)
{
	printf("[%d] ",piVector[i]);
}

printf("\r\n");

}
```

- 在第一部分中，定义了一个用于向量元素数目的符号常量 *VECTOR_SIZE*。
- 接下来，将定义一个 *piVector* 指针，该指针用于 int 数据类型的变量，并可使用 NULL 对其初始化。
- 接着，还将定义一个 i 计数器变量，该变量为 *int* 数据类型。
- 使用先前所创建的函数 *GetFilledVector()*，可创建一个用随机数填充的向量，其地址则存储在 *piVector* 指针中。于是，可对 *GetFilledVector()*函数的返回值的有效性进行检查。
- 所创建向量的各个元素均通过 *printf()*函数输出。此输出将在下一节显示。

诊断窗口中的输出

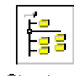
本节中所描述的实例在*诊断窗口*内生成下列输出结果：

```
Example 4
Vector Elements: [11478] [29358] [26962] [24464] [5705]
```

注意：
将结构传送给函数的过程和结构返回的过程均在下一章里描述。

4.7 结构

在 WinCC 项目 Project_C_Course 中，有关结构主题的实例可以通过单击如下所示的浏览栏图标来访问。实例在 cc_9_example_04.PDL 画面中组态。



Structures

结构类型的定义

除缺省数据类型以外，自定义的类型也可借助于结构来进行定义。在下列程序代码中，显示了一个主结构类型的定义。所定义的结构类型由一个 int 和一个 float 型结构元素组成。必须为每个结构元素分配一个名称。

```
struct ExampleStruct
{
	int iElement;
	float fElement;
};
```

结构变量的应用

在定义新的结构类型之后，即可定义 struct ExampleStruct 数据类型的变量。这已体现在下列程序代码中。同时，它还表明了如何访问结构变量的元素。

```
struct ExampleStruct* pesValue;

pesValue->iElement = 100;
pesValue->fElement = 2.5;
```

如果可利用的不是结构变量而只是一个指向结构变量的指针，则下列程序代码可以说明结构的单个元素是可以访问的。必须确保指针指向一个有效的结构变量或至少为其保留有存储空间。

4.7.1 实例 1 - 结构变量

在本实例中，对一个简单结构类型的定义和应用进行了解释。在*事件* → *鼠标* → *鼠标左击处*，为如下所示的对象*静态文本 1* 组态了本实例。



静态文本 1 的 C 动作

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszI
{

//define stucture "CC_POINT"
struct CC_POINT
{
       int iLeft;
       int iTop;
};

//define  structure tag "posObject"
struct CC_POINT posObject;

//set structure elements
posObject.iLeft = x - 8;
posObject.iTop = y - 8;

//access structure elements
SetLeft(lpszPictureName,"cool_man",posObject.iLeft);
SetTop(lpszPictureName,"cool_man",posObject.iTop);

}
```

- 在第一部分中，定义了 *CC_POINT* 结构类型，它由两个 int 元素组成。结构类型将接受鼠标单击的坐标。
- 接下来，将定义一个 *posObject* 结构变量，其数据类型为 *struct CC_POINT*。
- 接着，将值分配给 *posObject* 结构变量的元素。所分配的值即为鼠标单击的坐标。这些坐标可由 *C 动作*提供，它位于事件 → 鼠标 → 鼠标左击处，且以 x 和 y 的形式传送参数。
- 接下来，通过 SetLeft()和 SetTop()函数，使用结构变量所包含的值可对对象的坐标进行设置。

4.7.2 实例 2 - 类型定义

在本实例中，对简单结构类型的定义和应用进行了解释。与前述实例相反，该结构类型将用于整个项目中，而不仅仅是用于一种 *C 动作*。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 2* 组态了本实例。

Type Definition

apdefap.h 中的结构定义

```
#include "AP_PBIB.H"

//define structure tagCC_Rect
typedef struct tagCC_RECT
{
        int iLeft;
        int iTop;
        int iRight;
        int iBottom;
}
 CC_RECT,  //define type CC_RECT as struct tagCC_Rect
*PCC_RECT; //define type PCC_RECT as pointer to struct tagCC_Rect

//define constants for function GetFileName()
#define GFN_SAVE 0
#define GFN_OPEN 1
```

- 定义一个 *tagCC_RECT* 结构类型，它由四个 *int* 元素组成。该结构类型将接受一矩形区域的位置和尺寸。此结构类型的变量必须定义为一个 struct *tagCC_RECT* 数据类型的变量。为避免这种有时显得冗长的符号，可使用 *typedef* 语句来为其定义一个别名。如果现在就要定义一个这种数据类型的变量，则使用 *CC_RECT* 符号指定该数据类型就足够了。如果一个指针需要指向此种数据类型的变量，则需使用符号 *PCC_RECT*。

按钮 2 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

//define  and initialize CC_RECT structure
CC_RECT rect = { 10, 10, 20, 20};

//define  and initialize pointer to CC_RECT structure
PCC_RECT prect = NULL;

printf("\r\nExample 2\r\n");

//access struct elements
printf("Coordinates: %d, %d, %d, %d\r\n",
        rect.iLeft,rect.iTop,rect.iRight,rect.iBottom);

//access struct elements via pointer
prect = &rect;

printf("Coordinates: %d, %d, %d, %d\r\n",
        prect->iLeft,prect->iTop,prect->iRight,prect->iBottom);

}
```

- 在第一部分中，定义了一个 *CC_RECT* 数据类型的 *rect* 变量，并对其进行了初始化。
- 接下来，还定义一个 *PCC_RECT* 数据类型的 *prect* 变量，并使用 *NULL* 对其进行初始化。此数据类型是一个指向 *CC_RECT* 数据类型变量的指针。
- 接下来，通过操作符.访问结构变量 *rect* 的各个元素。通过 *printf()*函数可将其内容输出。
- 接下来，将 *rect* 变量的地址分配给 *prect* 指针。然后，通过操作符 ->访问 *prect* 指针当前所指向的结构变量的各个元素。通过 printf()函数可将结构变量的内容再次输出。下一节将显示此程序的输出。

诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出结果：

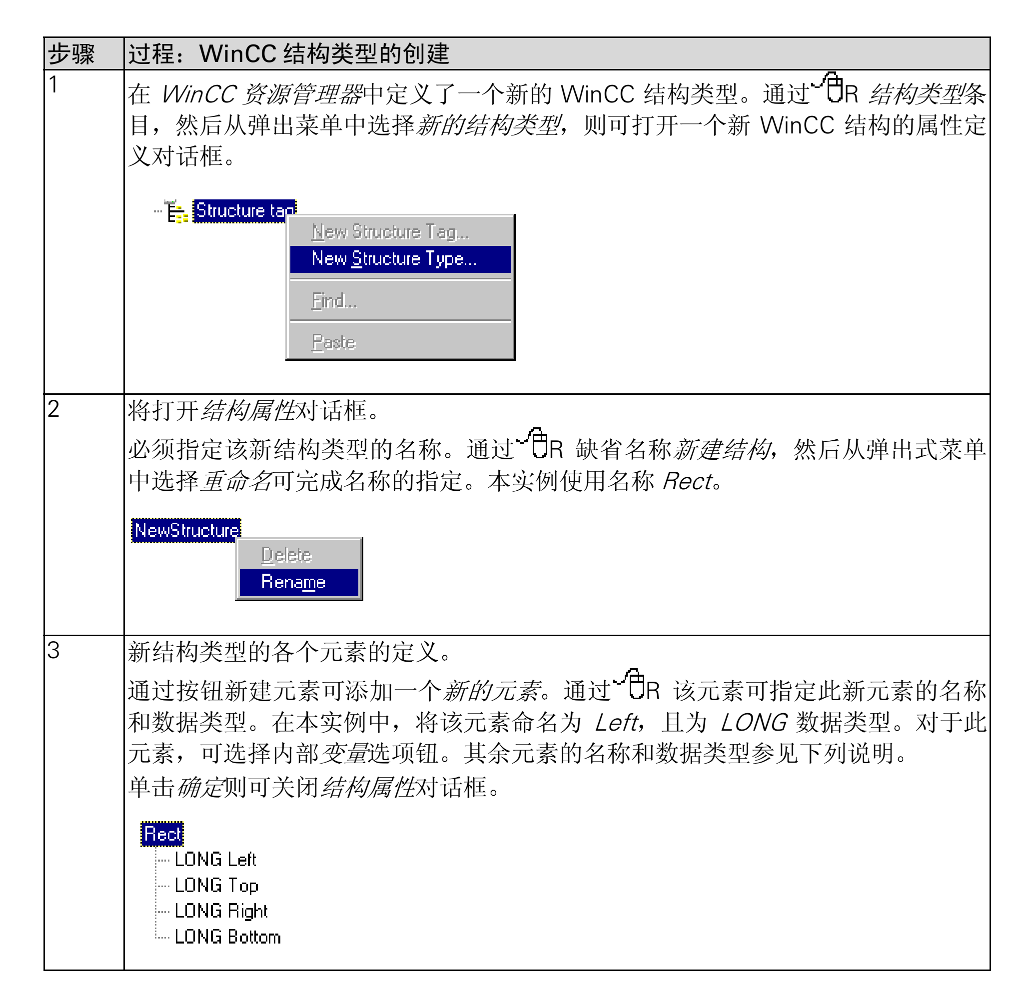
```
Example 2
Coordinates: 10, 10, 20, 20
Coordinates: 10, 10, 20, 20
```

4.7.3 实例 3 - WinCC 结构类型

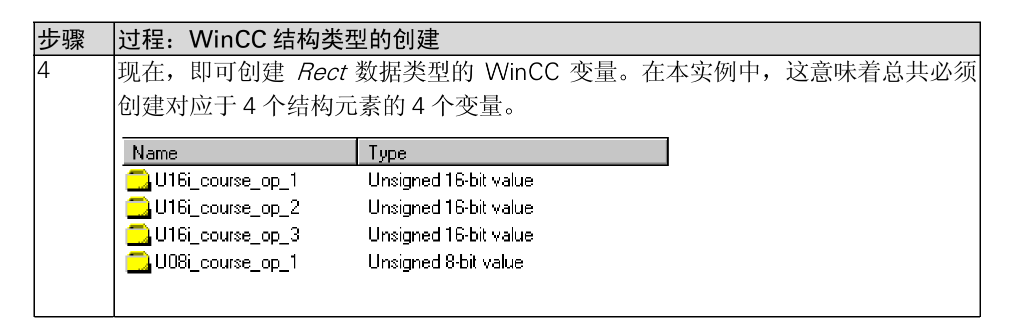
在此实例中，对 WinCC 结构类型的定义和应用进行了解释。其结构与前面实例中所定义的 *CC_RECT* 数据类型完全相同。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 3* 组态了本实例。



WinCC 结构类型的创建



步骤	过程：WinCC 结构类型的创建
1	在 *WinCC 资源管理器*中定义了一个新的 WinCC 结构类型。通过 R *结构类型*条目，然后从弹出菜单中选择*新的结构类型*，则可打开一个新 WinCC 结构的属性定义对话框。
2	将打开*结构属性*对话框。 必须指定该新结构类型的名称。通过 R 缺省名称*新建结构*，然后从弹出式菜单中选择*重命名*可完成名称的指定。本实例使用名称 *Rect*。
3	新结构类型的各个元素的定义。 通过按钮新建元素可添加一个*新的元素*。通过 R 该元素可指定此新元素的名称和数据类型。在本实例中，将该元素命名为 *Left*，且为 *LONG* 数据类型。对于此元素，可选择内部*变量*选项钮。其余元素的名称和数据类型参见下列说明。 单击*确定*则可关闭*结构属性*对话框。



步骤	过程：WinCC 结构类型的创建
4	现在，即可创建 *Rect* 数据类型的 WinCC 变量。在本实例中，这意味着总共必须创建对应于 4 个结构元素的 4 个变量。

Name	Type
U16i_course_op_1	Unsigned 16-bit value
U16i_course_op_2	Unsigned 16-bit value
U16i_course_op_3	Unsigned 16-bit value
U08i_course_op_1	Unsigned 8-bit value

按钮 3 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

//define  CC_RECT structure
CC_RECT rect;

//read element values of wincc structure tag
rect.iLeft = GetTagSDWord("STRi_course_str_1.Left");
rect.iTop = GetTagSDWord("STRi_course_str_1.Top");
rect.iRight = GetTagSDWord("STRi_course_str_1.Right");
rect.iBottom = GetTagSDWord("STRi_course_str_1.Bottom");

printf("\r\nExample 3\r\n");

//access struct elements
printf("Coordinates: %d, %d, %d, %d\r\n",
        rect.iLeft,rect.iTop,rect.iRight,rect.iBottom);

}
```

- 在第一部分中，定义了一个 *CC_RECT* 数据类型的 *rect* 变量。在前面的实例中已经定义了 *CC_RECT* 数据类型。
- 下一步，包含在某个 WinCC 结构变量中的值将被写入 *rect* 变量中的各个元素。在本实例中，通过 4 个 *I/O 域*可显示 WinCC 结构变量中所包含的数值，也可在其中对这些值进行编辑。
- 最后，通过 *printf()*函数可输出 *rect* 结构变量的各个元素。在下一节中将显示此程序的输出。

诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出结果：

Example 3
Coordinates: 0, 0, 0, 0

4.7.4 实例 4 - 读取 WinCC 结构类型的函数

在本实例中，将创建一个用来读取 WinCC 结构类型的函数，该函数已在前面的实例中进行了定义。然后，如同内部 GetTag 函数那样可以使用该函数。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 4* 组态了本实例。

Read Structure (GSC)

项目函数 GetTagRect()

```
#include "apdefap.h"
//include file which contains definition of structure tagCC_RECT

void* GetTagRect(char* lpszTagName)
{

PCC_RECT prect = NULL;

//max size of struct tag name = 260
//max size of element name = 7
char szElementTag[267];   //260 + 7

//allocate memory for CC_RECT structure
prect = SysMalloc(sizeof(CC_RECT));

//check return value of SysMalloc()
if (prect == NULL)
{
      return NULL;
}

///////////////////////////////////////////////////
//create tag names and get tag values
sprintf(szElementTag,"%s.Left",lpszTagName);
prect->iLeft = GetTagSDWord(szElementTag);

sprintf(szElementTag,"%s.Top",lpszTagName);
prect->iTop = GetTagSDWord(szElementTag);

sprintf(szElementTag,"%s.Right",lpszTagName);
prect->iRight = GetTagSDWord(szElementTag);

sprintf(szElementTag,"%s.Bottom",lpszTagName);
prect->iBottom = GetTagSDWord(szElementTag);
//
///////////////////////////////////////////////////

//return address of structure as void*
return (void*) prect;

}
```

- 在第一部分中，已集成了 *apdefap.h* 文件，该文件已包含了 *tagCC_RECT* 结构类型的定义。
- 在函数标题内，将此函数的名称指定为 *GetTagRect*。将字符串变量传送给该函数，该函数包含有将被读取的 WinCC 结构变量的名称。且将返回一个指针，该指针指向某个未定义数据类型(*void**)的存储块。
- 在下一部分中，将定义一个 *PCC_RECT* 数据类型的 *prect* 变量，并使用 *NULL* 对其进行初始化。该数据类型是一个指向 *CC_RECT* 数据类型变量的指针。

- 下一步，将创建一个字符串变量，用该变量接受 WinCC 结构变量的各个元素的名称。
- 接着，必须保留足够的内存空间来接受一个 *CC_RECT* 数据类型的变量。这可由内部函数 *SysMalloc()*来保证。所期望存储块的大小将传送给此函数，通过 *sizeof()*语句可确定该存储块的大小。如果可用的存储空间不够，则该函数将返回所保留的存储块的地址或 *NULL*。
- 接下来，将对 *SysMalloc()*函数发出的地址进行校验。如果没有足够的存储空间可供使用，则函数终止，并返回 *NULL*。
- 下一步，将组成 WinCC 结构变量的每一个元素的名称，并且对应元素的内容将被读入所保留的存储区域中。
- WinCC 结构变量的内容已经存放在所保留的存储块中，并且将返回该存储块的地址。该存储块将继续保留，即使该函数已退出，其数据也仍将保留。

注意：

如果仅仅是创建了一个 *CC_RECT* 数据类型的局部变量，而不是这里出现的程序，则该函数的调用者将接收到一个无效的指针，该局部变量的元素包含有 WinCC 结构变量的内容并将返回其地址。这可用如下事实来解释：在该函数结束处，*CC_RECT* 数据类型的变量将变为无效，因此返回一个无效对象的地址。

按钮 4 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

//define and initialize pointer to CC_RECT structure
PCC_RECT prect = NULL;

//read element values of wincc structure tag
prect = (PCC_RECT) GetTagRect("STRi_course_str_1");

printf("\r\nExample 4\r\n");

//check return value of GetTagRect()
if (prect == NULL)
{
	printf("\r\nError in GetTagRect()\r\n");
	return;
}

//access struct elements
printf("Coordinates: %d, %d, %d, %d\r\n",
	prect->iLeft,prect->iTop,prect->iRight,prect->iBottom);

}
```

- 第一步，将定义一个 *PCC_RECT* 数据类型的 *prect* 变量，并使用 *NULL* 对其进行初始化。
- 第二步，可通过前面创建的 *GetTagRect()*函数来读取 WinCC 结构变量。*GetTagRect()*函数将返回一个指针，该指针指向包含有所期望数据的存储块。该指针将转换为一个 *PCC_RECT* 类型的指针。
- 第三步，对从 *GetTagRect()*函数发出的指针进行检查。如果没有足够的存储空间可供使用，则该函数将返回数值 *NULL*。
- 第四步，将组成 WinCC 结构变量的每一个元素的名称，并将对应于各元素的内容读入所保留的存储区域中。
- WinCC 结构变量的内容已经存放在所保留的存储块中，将返回该存储块的地址。即使该函数退出之后，该内存块仍将继续保留，且其数据也将继续保存下来。
- 最后，通过 *printf()*函数可输出 *prect* 所指向的结构变量的各个元素。此程序的输出在下一部分中显示。

诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出结果：

```
Example 4
Coordinates: 0, 0, 0, 0
```

项目函数 SetTagRect()

除了 *GetTagRect()*函数以外，也可创建相对应的 *SetTagRect()*函数。在本实例中，为对 WinCC 结构变量进行初始化，可在 *cc_9_example_04.PDL* 画面的*事件 → 其它 → 打开画面处*使用该函数。

4.8 WinCC API

在 WinCC 项目 Project_C_Course 中，单击下面显示的浏览栏图标，可访问与主题 WinCC API 相关的实例。该实例可在 cc_9_example_10.PDL 画面中进行组态。



WinCC 应用程序编程接口

作为一种完全开放且可扩展的系统，WinCC 提供了一种广义的 API (应用程序编程接口)。这是一种供应用程序访问 WinCC 的接口。在 WinCC 项目本身中也可使用 WinCC API 的函数。
WinCC ODK (开放式开发工具包)给出了 WinCC API 的详细说明。其中，借助于函数描述和实例，对 WinCC API 进行了完整地解释。它还包括所有的头文件和必需的函数声明。然而，WinCC ODK 并非 WinCC 基本软件包的一部分，它必须单独购买。

函数库

WinCC 的每个(主要的)应用程序(图形编辑器、变量记录、报警记录等)都提供了其自己的 API，并位于一个或多个 DLL 中。DLL (动态装载库)是一个可动态装载的函数库。在关联的头文件中，将提供 DLL 所包含的函数声明。

在下面的程序代码实例中，将显示如何将 DLL 集成到 C 动作或其它函数中。在第一行里，指定了将要装载的 DLL 的名称。在此实例中，这是包含图形编辑器的 CS 函数的 DLL。在第二行中，将集成带有函数声明的头文件。如果只需要一个或两个函数，也可以直接在这里进行函数声明。可用#pragma code()来构成结束行。在上述实例中，DLL 与头文件二者的名称一致，是有一定道理的。然而，情况并不总是这样。

```
#pragma code("PDLCSAPI.dll")
#include "pdlcsapi.h"
#pragma code()
```

RT 函数和 CS 函数

每个应用程序的 API 函数可粗略地分为两种不同的类型。这就是所谓的 CS 函数(组态系统)和 RT 函数(运行系统)。

在大多数情况下，不用在 WinCC 项目中加载特定的 DLL 即可调用 RT 函数。RT 函数仅仅影响运行系统。在一个项目重新起动或即使在大多数情况下某画面改变之后，使用 RT 函数进行修改将会导致所做过的修改丢失。

在 WinCC 项目中应用 CS 函数之前，必须装载与之相应的 DLL，在其中已编入了函数。然而，在 WinCC 项目本身中应用 CS 函数仅在极少的情况下才具有意义。不过，当前的实例将足以阐明 CS 函数的应用，由此，可为自己的项目推演出其应用的基本原理。

实例项目

在此实例项目中，没有提供每一个 WinCC 应用程序 API 的详细描述。然而，使用图形编辑器 API 作为例子，对应用 WinCC API 进行工作的基本原理进行了解释。实例应用了 cc_9_example_10x.PDL 画面中的对象进行工作，它是为此目的而专门组态的。通过指定给本章的画面中的画面窗口显示出来。

4.8.1 实例 1 - 通过 RT 函数修改属性

此实例将对*图形编辑器* API 中的 RT 函数在对象属性的设置方面的应用进行说明。通过设置属性*位置 X* 和*位置 Y* 可对对象的位置进行修改。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 1* 组态了本实例。



按钮 1 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

BOOL bRet = FALSE;

//picture name without extension ".PDL"
char szPictureName[] = "cc_9_example_10ex";
char szObjectName[] = "I/OField1";

//property type
VARTYPE vt = VT_I4;
//new property values
int iValueLeft = 60;
int iValueTop  = 70;

//error structure
CMN_ERROR Error;

//////////////////////////////////////////////////////////////
//set the propertys and check the return values
bRet = PDLRTSetPropEx(0,szPictureName,szObjectName,"Left",
             vt,&iValueLeft,NULL,NULL,0,NULL,&Error);
if (bRet == FALSE)
{
      printf("\r\nError in PDLRTSetPropEx()\r\n"
             "\t%s\r\n",Error.szErrorText);
}

bRet = PDLRTSetPropEx(0,szPictureName,szObjectName,"Top",
             vt,&iValueTop,NULL,NULL,0,NULL,&Error);
if (bRet == FALSE)
{
      printf("\r\nError in PDLRTSetPropEx()\r\n"
             "\t%s\r\n",Error.szErrorText);
}
//
//////////////////////////////////////////////////////////////

}
```

- 在第一部分中，定义并初始化一个数据类型为 *BOOL* 的 *bRet* 变量。该变量将接受所调用的 API 函数的返回值。
- 接下来，定义了两个字符串变量。它们的内容(画面名称和对象名称)指定了将要编辑的对象。请确保画面名称中不包含文件扩展名 *PDL*。

- 为了对将要设置的属性类型进行定义，需要创建一个变量。为此，可以使用一个单独的数据类型。这就是 *VARTYPE* 数据类型。对每个已存在的属性类型均定义一个符号常量。本实例中将要设置的属性是 *VT_I4* 数据类型(具有 4 个字节长度的 *int* 类型)。
- 对于将要设置的*位置 X* 与*位置 Y* 属性，可为每个 *int* 类型定义一个变量，它包含属性的新值。
- 然后，定义了一个 *CMN_ERROR* 数据类型的变量。如果函数调用失败，则此结构将包括所产生错误的信息。
- 通过 API 函数 *PDLRTSetPropEx()*，可设置指定对象的*位置 X* 与*位置 Y* 属性。API 函数的第一个参数指定了对象的寻址模式。接下来的三个参数指定了所期望的画面名称、对象名称和属性名称。为指定所期望的属性，必须使用英文名称，而不是德文名称。对于前面的实例，这些就是 *Left* 与 *Top*。下一个参数表示属性类型。在下列参数中，指定了变量地址，它包含属性的新值。接下来的四个参数与所期望的功能无关。在最后一个参数中，指定了错误结构的地址。

  在调用了每个 API 函数后，使用 *if* 语句对其返回值进行检查。如果调用失败，则通过输出可对其进行说明。该输出中将有一部分包含了在错误结构的 *szErrorText* 结构元素中所包含的信息。

注意：
在本实例中，将 API 函数的返回值分配给一个 *BOOL* 数据类型的变量。然后，对该变量进行检查。API 函数的调用与对返回值的检查也可合并为一行。这种方法也将用于本章后面的实例中。

4.8.2 实例 2 - 通过 RT 函数创建一个变量连接

此实例将对*图形编辑器* API 中的 RT 函数应用于变量连接的创建进行说明。在 *I/O 域*中的*属性* → *输出/输入* → *输出值*处创建变量连接。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 2* 组态了本实例。

Tag Connection

按钮 2 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{
//include file with the LinkType definitions
#include "trigger.h"

char szPictureName[] = "cc_9_example_10ex";
char szObjectName[] = "I/OField1";

LINKINFO link;

CMN_ERROR Error;

//fill link info structure
link.LinkType = BUBRT_LT_VARIABLE_DIRECT;
link.dwCycle = 0;
strcpy(link.szLinkName,"U08i_course_wincc_2");

//set link and check the return value
if ( PDLRTSetLink(0,szPictureName,szObjectName,"OutputValue",
                  &link,NULL,NULL,&Error) == FALSE)
{
      printf("\r\nError in PDLRTSetLink()\r\n%s\r\n",
             Error.szErrorText);
}

}
```

- 在第一部分中，集成 *trigger.h* 文件。该文件包含了本实例中所使用的符号常量的定义。
- 接下来，定义了两个字符串变量。它们的内容(画面名称和对象名称)指定了将要编辑的对象。
- 为了指定变量连接属性，可使用一个单独的数据类型。这就是 *LINKINFO* 结构类型。定义一个 *LINKINO* 数据类型的 link 变量。
- 接下来，定义了一个 *CMN_ERROR* 数据类型的变量。
- *link* 变量的结构元素可使用所期望的变量连接的信息来填充。为 *LinkType* 元素分配 *BUBRT_LT_VARIABLE_DIRECT* 符号常量。该常量代表一个直接变量连接。为 *dwCycle* 元素分配数值 *0*，它对应于一旦修改触发器。*szLinkName* 元素指定了要使用的变量。

- 通过 API 函数 *PDLRTSetLink()*，可创建指定对象的*变量连接*。API 函数的第一个参数指定了对象的寻址模式。接下来的三个参数指定了所期望的画面名称、对象名称和属性名称。在下面的参数中，指定了 *link* 变量的地址，其确定了将要创建的变量连接。接下来的两个参数与所期望的功能无关。最后一个参数指定了错误结构的地址。如果调用 API 函数失败，则通过输出可对其进行说明。

注意：
本章的前两个实例有关 *cc_9_example_10x.PDL* 画面的 *I/O 域 1* 对象。第一个实例改变了对象在画面中的位置，第二个实例在其上创建了一个变量连接。在一幅画面改变以后，所做的修改将会丢失。请确保在后面实例中所描述的 *CS* 函数执行之后，必须总是执行画面的修改。也就是说单击前两个按钮以外的任一按钮后，使用前两个按钮所完成的画面修改将会丢失。

4.8.3 实例 3 - 通过 CS 函数创建新对象

在此实例中，将显示如何应用*图形编辑器* API 中的 CS 函数创建一个新对象。将创建一个新的 *I/O 域*。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 3* 组态了本实例。

New I/O Field

按钮 3 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper·
{
//PDLCSAPI.dll contains the graphics designer cs functions
#pragma code("PDLCSAPI.dll")
#include "pdlcsapi.h"
#pragma code()
//include file with the GUID definitions
#include "pdl_guid.h";

char szProjectName[_MAX_PATH];
char szPictureName[] = "cc_9_example_10ex.PDL";//with ".PDL"
DWORD dwFlags = 1;//do not display picture
CMN_ERROR Error;
char szObjectName[] = "I/OField2";
GUID guid = GUID_IOField;//object type i/o field

//get project name
if ( DMGetRuntimeProject(szProjectName,_MAX_PATH+1,&Error) == FALSE )
{
      printf("\r\nError in DMGetRuntimeProject()\r\n"
             "\t%s\r\n",Error.szErrorText);
      return;
}
//initialize API interface of the graphics designer
if ( PDLCSGetOleAppPtr(FALSE,&Error) == FALSE )
{
      printf("\r\nError in PDLCSGetOleAppPtr()\r\n%s\r\n",
             Error.szErrorText);
      return;
}
//open picture without displaying it
if ( PDLCSOpenEx(szProjectName,szPictureName,dwFlags,&Error) == FALSE )
{
      printf("\r\nError in PDLCSOpenEx()\r\n%s\r\n",
             Error.szErrorText);
      goto OPEN_FAILED;
}
//create object
if ( PDLCSNewObjectEx(szProjectName,szPictureName,&guid,
                        szObjectName,&Error) == FALSE )
{
      printf("\r\nError in PDLCSNewObjectEx()\r\n%s\r\n",
      Error.szErrorText);
      goto ACTION_FAILED;
}
//save picture
if ( PDLCSSave(szProjectName,szPictureName,&Error) == FALSE )
{
      printf("\r\nError in PDLCSSave()\r\n%s\r\n",
      Error.szErrorText);
      goto ACTION_FAILED;
}
//actualize the picture which contains the created object
ActualizeObjects();
//close picture
ACTION_FAILED: PDLCSClose(szProjectName,szPictureName,&Error);
//disconnect from the API interface of the graphics designer
OPEN_FAILED:   PDLCSDelOleAppPtr(FALSE);
}
```

- 在第一部分中，装载*图形编辑器* API 的 DLL。并集成 *pdl_guid.h* 文件，它包含了本实例所使用的符号常量的定义。
- 接下来，将定义一个 *szProjectName* 字符串变量，用于接受项目名称。
- 然后再定义一个字符串变量，用于接受画面名称。请注意：画面名称必须具有文件扩展名 *PDL* (以区分 RT 函数)。
- 接着定义所需要的其它变量。其中包括一个 *GUID* 数据类型的变量，它确定了将要创建的对象类型。
- 在下一部分中，将通过 API 函数 *DMGetRuntimeProject()*来确定项目名称。该项目名称将存储在 *szProjectName* 变量中。如果未能确定项目名称，则将通过输出对其进行说明。在这种情况下，将通过返回语句退出 *C 动作*。
- 在下一部分中，将通过 API 函数 *PDLCSGetOleAppPtr()*来对*图形编辑器* API 进行初始化。如果图形编辑器 API 的初始化失败，则通过输出对其进行说明。在这种情况下，也将通过返回语句来退出 *C 动作*。
- 在下一部分中，通过 API 函数 *PDLCSOpenEx()*打开将要编辑的画面。从下一个参数到最后一个参数，将 *dwFlags* 变量设置为数值 1 并传送给 API 函数。这将使画面不会在屏幕上显示。如果未能打开画面，则将通过输出对其进行说明。通过 *goto* 语句，可跳转到与*图形编辑器* API 终止连接的代码位置。
- 在下一部分中，通过 API 函数 *PDLCSNewObjectEx()*创建一个名为 *I/O 域 2* 的新对象。如果未能创建新对象，则将通过输出对其进行说明。通过 *goto* 语句，将跳转到先前打开的画面所关闭的代码位置。
- 在下一部分中，可通过 API 函数 *PDLCSSave()*对画面进行保存。如果未能保存画面，则将通过输出对其进行说明。通过 *goto* 语句，将跳转到先前打开的画面所关闭的代码位置(如上一部分)。
- 接下来，通过*项目函数 ActualizeObjects()*再次选择要编辑的画面。
- 下一步，通过 API 函数 *PDLCSClose()*再次关闭先前已打开的画面。在该语句之前插入了一个符号，它就是先前的 *goto* 语句的跳转目标。
- 最后，通过 API 函数 *PDLCSDelOleAppPtr()*再次终止与*图形编辑器* API 的连接。在该语句之前也插入了一个符号，它就是先前的 *goto* 语句的跳转目标。

注意：
将在后面实例中创建的 *C 动作*与本实例中所创建的 *C 动作*极为相似。为此，我们对在本实例中已作过的详细解释将不再赘述。有关的代码描述将仅限于对程序的执行进行概述。

4.8.4 实例 4 - 通过 CS 函数修改属性

此实例将对*图形编辑器* API 的 CS 函数应用于对象属性的设置进行说明。通过设置属性*位置 X* 和*位置 Y* 可对对象的位置进行修改。在*事件 → 鼠标 → 鼠标动作*处为如下所示的对象*按钮 4* 组态了本实例。



按钮 4 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{
#pragma code("PDLCSAPI.dll")
#include "pdlcsapi.h"
#pragma code()

char szProjectName[_MAX_PATH];
char szPictureName[] = "cc_9_example_10ex.PDL";
char szObjectName[] = "I/OField2";
char szPropertyName[2][5] = { "Left", "Top" };
VARTYPE vt = VT_I4;
int iValue[] = { 60, 130 };
int i;
CMN_ERROR Error;
//get project name
if (DMGetRuntimeProject(szProjectName,_MAX_PATH+1,&Error) == FALSE)
{
      printf("\r\nError in DMGetRuntimeProject()\r\n"
             "\t%s\r\n",Error.szErrorText);
      return;
}
//initialize API interface of the graphics designer
if ( PDLCSGetOleAppPtr(FALSE,&Error) == FALSE)
{
      printf("\r\nError in PDLCSGetOleAppPtr()\r\n%s\r\n",
             Error.szErrorText);
      return;
}
//open picture without displaying it
if ( PDLCSOpenEx(szProjectName,szPictureName,1,&Error) == FALSE)
{
      printf("\r\nError in PDLCSOpenEx()\r\n%s\r\n",
             Error.szErrorText);
      goto OPEN_FAILED;
}
//set propertys
for (i=0; i<2; i++)
{
      if (PDLCSSetPropertyEx(szProjectName,szPictureName,szObjectName,
                 szPropertyName[i],vt,iValue+i,0,NULL,&Error) == FALSE)
      {
            printf("\r\nError in PDLCSSetPropertyEx()\r\n%s\r\n",
            Error.szErrorText);
      }
}
//save picture
if ( PDLCSSave(szProjectName,szPictureName,&Error) == FALSE)
{
      printf("\r\nError in PDLCSSave()\r\n%s\r\n",
      Error.szErrorText);
      goto ACTION_FAILED;
}
//actualize the picture which contains the created object
ActualizeObjects();
//close picture
ACTION_FAILED: PDLCSClose(szProjectName,szPictureName,&Error);
//disconnect from the API interface of the graphics designer
OPEN_FAILED:   PDLCSDelOleAppPtr(FALSE);
}
```

- 在第一部分中，将装载*图形编辑器* API 的 DLL。
- 在第二部分中，定义了所需要的变量。将要设置的属性名称及属性值均存储在向量中，与实例 1 中所描述的过程相对应。
- 通过 API 函数 *DMGetRuntimeProject()*可确定项目名称。
- 通过 API 函数 *PDLCSGetOleAppPtr()*可对图形编辑器 API 进行初始化。
- 通过 API 函数 *PDLCSOpenEx()*来打开要编辑的画面。
- 在 *for* 循环中，可通过 API 函数 *PDLCSSetPropertyEx()*来设置对象属性。如果不同类型的属性均按这种方式进行设置，则必须定义一个向量，而不是 *vt* 变量。该向量将确定所要设置的每个属性的属性类型。
- 可通过 API 函数 *PDLCSSave()*对画面进行保存。
- 通过项目函数 *ActualizeObjects()*再次选择所要编辑的画面。
- 通过 API 函数 *PDLCSClose()*再次关闭先前已打开的画面。
- 通过 API 函数 *PDLCSDelOleAppPtr()*再次终止与*图形编辑器* API 的连接。

4.8.5 实例 5 - 通过 CS 函数创建一个变量连接

此实例将对*图形编辑器* API 中的 CS 函数应用于变量连接的创建进行说明。在 *I/O 域的属性 → 输出/输入→ 输出值*里创建变量连接。在*事件 → 鼠标 → 鼠标动作*处为如下所示的对象*按钮 5* 组态了本实例。

Tag Connection

按钮 5 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{
#pragma code("PDLCSAPI.dll")
#include "pdlcsapi.h"
#pragma code()

char szProjectName[_MAX_PATH];
char szPictureName[] = "cc_9_example_10ex.PDL";
char szObjectName[] = "I/OField2";
LINK_INFO link;
CMN_ERROR Error;

//get project name
if (DMGetRuntimeProject(szProjectName,_MAX_PATH+1,&Error) == FALSE)
{
      printf("\r\nError in DMGetRuntimeProject()\r\n"
             "\t%s\r\n",Error.szErrorText);
      return;
}
//initialize API interface of the graphics designer
if ( PDLCSGetOleAppPtr(FALSE,&Error) == FALSE)
{
      printf("\r\nError in PDLCSGetOleAppPtr()\r\n%s\r\n",
             Error.szErrorText);
      return;
}
//open picture without displaying it
if ( PDLCSOpenEx(szProjectName,szPictureName,1,&Error) == FALSE)
{
      printf("\r\nError in PDLCSOpenEx()\r\n%s\r\n",
             Error.szErrorText);
      goto OPEN_FAILED;
}
//set link info struct
link.enumLinkType = BUBRT_LT_VARIABLE_DIRECT;
link.dwCycle = 0;
strcpy(link.szLinkName,"U08i_course_wincc_1");
//set link
if ( PDLCSSetLink(szProjectName,szPictureName,szObjectName,"OutputValue",
             &link,&Error) == FALSE)
{
      printf("\r\nError in PDLCSSetLink()\r\n%s\r\n",
             Error.szErrorText);
      goto ACTION_FAILED;
}
//save picture
if ( PDLCSSave(szProjectName,szPictureName,&Error) == FALSE)
{
      printf("\r\nError in PDLCSSave()\r\n%s\r\n",
      Error.szErrorText);
      goto ACTION_FAILED;
}
//actualize the picture which contains the created object
ActualizeObjects();
//close picture
ACTION_FAILED: PDLCSClose(szProjectName,szPictureName,&Error);
//disconnect from the API interface of the graphics designer
OPEN_FAILED:   PDLCSDelOleAppPtr(FALSE);
}
```

- 在第一部分中，装载*图形编辑器* API 的 DLL。
- 在第二部分中，定义了所需要的变量。
- 通过 API 函数 *DMGetRuntimeProject()*可确定项目名称。
- 通过 API 函数 *PDLCSGetOleAppPtr()*可对图形编辑器 API 进行初始化。
- 通过 API 函数 *PDLCSOpenEx()*打开所要编辑的画面。
- 在下一部分中，*link* 变量的结构元素可使用所期望的变量连接信息来填充。
- 通过 *PDLCSSetLink()*函数创建对象的变量连接。
- 通过 API 函数 *PDLCSSave()*对画面进行保存。
- 通过项目函数 *ActualizeObjects()*再次选择所要编辑的画面。
- 通过 API 函数 *PDLCSClose()*再次关闭先前已打开的画面。
- 通过 API 函数 *PDLCSDelOleAppPtr()*再次终止与*图形编辑器* API 的连接。

4.8.6 实例 6 - 通过 CS 函数列出对象

此实例将对*图形编辑器* API 中的 CS 函数用来列出画面中所包含的对象进行说明。对于每个可用对象，API 将调用一个专门创建的函数，并为其传送相应对象的有关信息。这样的函数可称之为回调函数。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 6* 组态了本实例。

List Objects

项目函数 ObjectCallback()

```
#include "pdlcsapi.h"
//include file with OBJECT_INFO_STRUCT definition

BOOL ObjectCallback(void* lpData,void* item)
{

//pointer to OBJECT_INFO_STRUCT
LPOBJECT_INFO_STRUCT lpInfoStruct = NULL;

//store received address of OBJECT_INFO_STRUCT
lpInfoStruct = (LPOBJECT_INFO_STRUCT) lpData;

//check received address
if (lpInfoStruct == NULL)
{
      printf("Error in ObjectCallback()\r\n");
      return FALSE;
}

//access structure element
printf("%s\r\n",lpInfoStruct->szObjectName);

return TRUE;

}
```

- 在第一部分中，集成了 *pdlcsapi.h* 文件，该文件包含了 *OBJECT_INFO_STRUCT* 结构类型的定义。
- 还为该函数指定了返回值的数据类型以及传送参数的数据类型和数量，这些均可从 WinCC ODK 中获得。
- 接下来，定义一个指向 *OBJECT_INFO_STRUCT* 数据类型的变量指针。第一个传送参数中所包含的地址将分配给该指针。然后，对该指针的有效性进行检查。
- 将接收了 *OBJECT_INFO_STRUCT* 的对象的名称输出。

按钮 6 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

#pragma code("PDLCSAPI.dll")
#include "pdlcsapi.h"
#pragma code()

char szProjectName[_MAX_PATH];
char szPictureName[] = "cc_9_example_10ex.PDL";
CMN_ERROR Error;

//get project name
if ( DMGetRuntimeProject(szProjectName,_MAX_PATH+1,&Error) == FALSE)
{
      printf("\r\nError in DMGetRuntimeProject()\r\n"
             "\t%s\r\n",Error.szErrorText);
      return;
}
//initialize API interface of the graphics designer
if ( PDLCSGetOleAppPtr(FALSE,&Error) == FALSE)
{
      printf("\r\nError in PDLCSGetOleAppPtr()\r\n%s\r\n",
             Error.szErrorText);
      return;
}
//open picture without displaying it
if ( PDLCSOpenEx(szProjectName,szPictureName,1,&Error) == FALSE)
{
      printf("\r\nError in PDLCSOpenEx()\r\n%s\r\n",
             Error.szErrorText);
      goto OPEN_FAILED;
}
printf("\r\nObjects in picture cc_9_example_10ex.PDL:\r\n");
//enumerate objects
if ( PDLCSEnumObjList(szProjectName,szPictureName,
                      ObjectCallback,NULL,&Error) == FALSE)
{
      printf("\r\nError in PDLCSEnumObjectList()\r\n%s\r\n",
             Error.szErrorText);
}
//close picture
PDLCSClose(szProjectName,szPictureName,&Error);
//disconnect from the API interface of the graphics designer
OPEN_FAILED:   PDLCSDelOleAppPtr(FALSE);

}
```

- 在第一部分中，装载*图形编辑器* API 的 DLL。
- 在第二部分中，定义了所需要的变量。
- 通过 API 函数 *DMGetRuntimeProject()*可确定项目名称。
- 通过 API 函数 *PDLCSGetOleAppPtr()*可对*图形编辑器* API 进行初始化。
- 通过 API 函数 *PDLCSOpenEx()*打开要编辑的画面。
- 包含在画面中的所有对象均将通过 API 函数 *PDLCSEnumObjList()*列出。为此，先前创建的 *ObjectCallback()*项目函数的地址将传送给 API 函数。这种类型的函数也称之为回调函数。画面中包含的每个对象均将调用该函数一次，调用时将传送一个对象的数据。
- 通过 API 函数 *PDLCSClose()*再次关闭先前已打开的画面。
- 通过 API 函数 *PDLCSDelOleAppPtr()*再次终止与图形编辑器 API 的连接。

诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出：

```
Objects in picture cc_9_example_10ex.PDL:
cc_9_example_10ex
I/OField1
I/OField2
```

4.9 项目环境

在 WinCC 项目 Project_C_Course 中，单击如下显示的浏览栏图标，可访问与主题项目环境有关的实例。可在 cc_9_example_11.PDL 画面中组态该实例。


Project Environment

常规信息

在很多情况下，编制 C 动作或其它函数均需要对文件路径、本地计算机名称等进行详细说明。然后，根据当前环境，将这些值指定为绝对值。如果将项目传送给另外一台计算机，则可能会出现问题。这里所遇到的环境完全不同于创建系统中的环境。因此，建议不要使用绝对路径进行说明，在创建一个项目时，尤其如此。在运行系统中应确定这类信息。本章所包含的实例说明了如何访问与本地计算机上的环境相关的信息。为此，将使用 WinCC API 和 Windows API。

4.9.1 实例 1 - 项目文件的确定

本实例概述了 WinCC 项目中的项目文件的确定过程。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 1* 组态了本实例。

Project File

按钮 1 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

BOOL bRet;
char szProjectFile[_MAX_PATH+1];
CMN_ERROR Error;

//get the project file *.mcp
bRet = DMGetRuntimeProject(szProjectFile,_MAX_PATH+1,&Error);

//check return value
if (bRet == FALSE)
{
      printf("\r\nError in DMGetRuntimeProject()\r\n"
             "\t%s\r\n",Error.szErrorText);
      return;
}

//display project file
printf("\r\nProjectFile:\r\n%s\r\n",szProjectFile);

}
```

- 在第一部分中，定义了一个数据类型为 *BOOL* 的 *bRet* 变量。
- 下一步，将定义一个 *szProjectFile* 字符串变量，用于接受项目名称。将该变量的长度设置为可容纳(存储)最长的路径说明。
- 接下来，定义了一个 *CMN_ERROR* 数据类型的变量。
- 再接下来，通过 API 函数 *DMGetRuntimeProject()*确定项目名称。该项目名称将存储在 *szProjectFile* 变量中。在第二个参数中指定了为项目名称所保留的存储空间的大小。在第三个参数中指定了错误结构的地址。如果不需要任何出错信息，则可传送 *NULL*。
- 接下来，检查 API 函数 *DMGetRuntimeProject()*的返回值。
- 最后，输出已确定的项目文件名称。此输出在下一部分中显示。

诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出：

ProjectFile:
\\ZIP-WS5\WinCC50_Project_Project_C_Course\Project_C_Course.MCP

4.9.2 实例 2 - 确定项目路径

本实例概述了 WinCC 项目中的项目路径的确定过程。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 2* 组态了本实例。

Project Path

按钮 2 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{

BOOL bRet = FALSE;
char szProjectFile[_MAX_PATH+1];
char* psz = NULL;
CMN_ERROR Error;

//get the project file *.mcp
bRet = DMGetRuntimeProject(szProjectFile,_MAX_PATH+1,&Error);

//check return value
if (bRet == FALSE)
{
	printf("\r\nError in DMGetRuntimeProject()\r\n"
		"\t%s\r\n",Error.szErrorText);
	return;
}

//search for last backslash
psz = strrchr(szProjectFile,'\\');

//cut string after last backslash
if (psz != NULL)
{
	*(psz+1) = 0;
}

//display project path
printf("\r\nProjectPath:\r\n%s\r\n",szProjectFile);

}
```

- 在第一部分中，定义并初始化一个数据类型为 *BOOL* 的 *bRet* 变量。
- 下一步，定义了一个 *szProjectFile* 字符串变量，用于接受项目名称。此外，将字符串变量定义为 *char** 类型，并使用 *NULL* 对其进行初始化。
- 接下来，定义一个 *CMN_ERROR* 数据类型的变量。
- 通过 API 函数 *DMGetRuntimeProject()* 可确定项目的名称。
- 接下来，检查 API 函数 *DMGetRuntimeProject()* 的返回值。
- 下一步，*strrchr()* 函数将对已确定的项目文件名称中处于“\”字符的最后位置进行搜索。在所发现字符后面的一个位置上，插入一个 0。仅将保留项目文件的路径说明，而不保留项目文件名称自身。
- 最后，输出已确定的项目路径。此输出在下一部分中显示。

诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出：

ProjectPath:
\\ZIP-WS5\WinCC50_Project_Project_C_Course

4.9.3 实例 3 - 通过项目函数确定项目路径

在本实例中，将在先前的实例中所说明的项目文件夹的确定转换给一个*项目函数*。*在事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 3* 组态了本实例。

Project Path (GSC)

项目函数 GetProjectPath()

```
BOOL GetProjectPath(char* lpstrProjectPath)
{
BOOL bRet = FALSE;
char szProjectFile[_MAX_PATH+1];
char* psz = NULL;
CMN_ERROR Error;

bRet = DMGetRuntimeProject(szProjectFile,_MAX_PATH+1,&Error);

if (bRet == FALSE)
{
      return FALSE;
}

psz = strrchr(szProjectFile,'\\');

if (psz == NULL)
{
      return FALSE;
}

*(psz+1) = 0;

strcpy(lpstrProjectPath,szProjectFile);

return TRUE;

}
```

- 将一个字符串变量传送给项目函数，已确定的项目路径将写入到该变量中。函数的调用者务必确保为该字符串变量保留了足够的存储空间。如果已成功地执行了函数，则可看到其返回值。
- 项目路径的确定过程与前一个实例中所说明的过程具有相同原理。
- 所确定的项目路径将通过 *strcpy()*函数复制给所传送的字符串变量。

按钮 3 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

BOOL bRet = FALSE;
char szProjectPath[_MAX_PATH+1];

//project function to get project path
bRet = GetProjectPath(szProjectPath);

//check return value
if (bRet == FALSE)
{
	printf("\r\nError in GetProjectPath()\r\n");
	return;
}

//display project path
printf("\r\nProjectPath:\r\n%s\r\n",szProjectPath);

}
```

- 在第一部分中，定义并初始化了一个数据类型为 *BOOL* 的 *bRet* 变量。
- 下一步，定义一个 *szProjectPath* 用于接受项目路径的字符串变量。
- 使用先前所创建的 *GetProjectPath()项目函数*来确定项目路径。随后，检查项目函数的返回值。
- 最后，输出已确定的项目路径。

4.9.4 实例 4 - 确定安装文件夹

在本实例中概述了确定 WinCC 的安装文件夹的过程。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 4* 组态了本实例。

Installation Folder

按钮 4 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

char szProjectFile[_MAX_PATH+1];
DM_DIRECTORY_INFO dmDirInfo;
CMN_ERROR Error;
char* psz = NULL;

//get the project file *.mcp
if ( DMGetRuntimeProject(szProjectFile,_MAX_PATH+1,&Error) == FALSE)
{
      printf("\r\nError in DMGetRuntimeProject()\r\n"
            "\t%s\r\n",Error.szErrorText);
      return;
}

//get wincc directories
if ( DMGetProjectDirectory("",szProjectFile,&dmDirInfo,&Error) == FALSE)
{
      printf("\r\nError in DMGetProjectDirectory()\r\n"
            "\t%s\r\n",Error.szErrorText);
      return;
}

if ( (psz = strrchr(dmDirInfo.szGlobalLibDir,'\\')) != NULL)
{
      *psz = 0;
}
if ( (psz = strrchr(dmDirInfo.szGlobalLibDir,'\\')) != NULL)
{
      *(psz+1) = 0;
}

//display installation directory
printf("\r\nInstallationDirectory:\r\n%s\r\n",dmDirInfo.szGlobalLibDir);

}
```

- 在第一部分中，定义用于接受项目文件名称的字符串变量 *szProjectFile*。
- 下一步，定义用于接受路径信息的变量 *dmDirInfo*。这是一个 *DM_DIRECTORY_INFO* 结构类型的变量。
- 接下来，定义一个 *CMN_ERROR* 数据类型的变量。
- 然后，定义一个 *char** 型字符串变量，并用 *NULL* 对其进行初始化。
- 通过 API 函数 *DMGetRuntimeProject()*确定项目的名称。

- 通过 API 函数 *DMGetProjectDirectory()*为变量 *dmDirInfo* 填充路径信息。变量中所包含的路径之一是全局库文件夹的路径。该路径存储在 *szGlobalLibDir* 结构元素中。全局库文件夹是 WinCC 安装文件夹的子文件夹。
- 使用第一个 strrchr()函数，在已确定的路径中搜索最后一个“\”字符，并用“0”来取代它。使用第二个 strrchr()函数，在其余路径中搜索最后一个“\”字符。在其后的一个位置上，插入一个 0。
- 最后，输出已确定的安装文件夹。此输出在下一部分中显示。

诊断窗口中的输出

本节中描述的实例在*诊断窗口*内生成下列输出：

```
InstallationDirectory:
C:\Siemens\WinCC\
```

4.9.5 实例 5 - 确定计算机名称

在本实例中概述了确定本地计算机名称的过程。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 5* 组态了本实例。

Computer Name

按钮 5 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

#pragma code ("Kernel32.DLL");
BOOL GetComputerNameA(LPSTR ComputerName, LPDWORD pdwSize);
#define MAX_COMPUTERNAME_LENGTH 15
#pragma code();

BOOL bRet = FALSE;
char szComputerName[MAX_COMPUTERNAME_LENGTH + 1];
DWORD dwSize = MAX_COMPUTERNAME_LENGTH + 1;

bRet = GetComputerNameA(szComputerName,&dwSize);

//check return value
if (bRet == FALSE)
{
      printf("\r\nComputerName:\r\nUnknown Computer\r\n");
      return;
}

//display project file
printf("\r\nComputerName:\r\n%s\r\n",szComputerName);

}
```

- 在第一部分中，集成 Windows DLL Kernel32。由于只需要 DLL 的一个函数，因此直接声明该函数。此外，还将定义一个符号常量，用于记录计算机名称的最大长度。
- 下一步，定义并初始化一个 *BOOL* 数据类型的变量 *bRet*。
- 然后，定义一个字符串变量 *szComputerName*，用于接受计算机名称。此外，还定义一个 *DWORD* 数据类型的变量，并用先前所定义的字符串变量的长度对其进行初始化。
- 通过 Windows 函数 *GetComputerNameA()*确定本地计算机的名称。该名称被写入所传送的字符串变量 *szComputerName* 中。
- 接着，检查 Windows 函数 *GetComputerNameA()*的返回值。
- 最后，输出已确定的计算机名称。

4.9.6 实例 6 - 确定用户名

在本实例中概述了确定当前登录 *Windows NT* 的用户的过程。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 6* 组态了本实例。

User Name

按钮 6 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

#pragma code ("advapi32.DLL");
BOOL GetUserNameA(LPSTR UserName, LPDWORD pdwSize);
#define UNLEN 256
#pragma code();

BOOL bRet = FALSE;
char szUserName[UNLEN + 1];
DWORD dwSize = UNLEN + 1;

bRet = GetUserNameA(szUserName,&dwSize);

//check return value
if (bRet == FALSE)
{
      printf("\r\nUserName:\r\nUnknown User\r\n");
      return;
}

//display project file
printf("\r\nUserName:\r\n%s\r\n",szUserName);

}
```

- 在第一部分中，集成 Windows DLL advapi32。由于只需要 DLL 的一个函数，因此直接声明该函数。此外，还定义一个符号常量，用于记录用户名的最大长度。
- 下一步，定义并初始化一个 *BOOL* 数据类型的变量 *bRet*。
- 然后，定义一个字符串变量 *szUserName*，用于接受用户名。此外，还定义一个 *DWORD* 数据类型的变量，并用先前所定义的字符串变量的长度对其进行初始化。
- 通过 Windows 函数 *GetUserNameA()*确定当前登录到 Windows NT 上的用户的名称。该名称被写入所传送的字符串变量 *szUserName* 中。
- 接着，检查 Windows 函数 *GetUserNameA()*的返回值。
- 最后，输出已确定的用户名。

4.10 Windows API

本章所描述的实例在 WinCC 项目 Project_C_Course 中的画面 cc_0_startpicture_00.PDL 与 cc_2_keyboard_01.PDL 中进行组态。

Windows 应用程序接口

除 WinCC API 之外，在 WinCC 项目中也可以使用所有的 Windows API。这使得几乎可以不受限制地对系统进行访问。

下列实例可使您对该主题有一个概略的了解。通过这些实例说明了 Windows API 的一般应用过程。然而，这并非有关 Windows API 的详尽叙述。

Windows API 的函数也位于不同的 DLL 中，就如 WinCC API 的函数一样。在各种不同的头文件中对这些函数进行了声明。DLL 的集成遵循集成 WinCC DLL 所使用的同一原理。下列实例程序代码对这种集成进行了说明。

```
#pragma code ("comdlg32.dll")
#include "commdlg.h"
#pragma code()
```

4.10.1 实例 1 - 设置 Windows 属性

本实例说明如何更改 Windows 窗口的属性。在本实例中，更改运行系统窗口的标题和几何结构。该实例在起始画面 *cc_0_startpicture_00.PDL* 的*事件* → *其它* → *打开画面处*进行组态。



起始画面的 C 动作

```
#include "apdefap.h"
void OnOpenPicture(char* lpszPictureName, char* lpszObjectName, char* lpszI
{

//get handle of runtime window
HWND hWnd = NULL;
hWnd = FindWindow(NULL,"WinCC-Runtime - ");

//set text of runtime window
SetWindowText(hWnd,"WinCC C-Course");
//set position and size of runtime window
SetWindowPos(hWnd,HWND_TOP,0,0,1024,768,0);

//set active the first chapter
SetTagByte("U08i_org_bar_1",0);

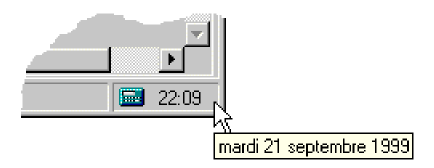
//
CreateExternalTags();

}
```

- 本实例中所使用的 Windows 函数在 WinCC 项目中是已知的。因此，不需要加载任何 Windows DLL。
- 在第一部分中，定义一个 *HWND* 类型的变量，并用 *NULL* 对其进行初始化。该变量就是所谓的窗口句柄，即指向某个 Windows 窗口的指针。
- 通过 Windows 函数 *FindWindow()*，可借助指定窗口标题来确定一个 Windows 窗口的窗口句柄。如果已指明运行系统窗口的缺省标题，就可以确定其窗口句柄。
- 通过 Windows 函数 *SetWindowText()*，可更改运行系统窗口的标题。在本实例中，将标题更改为 *WinCC C-Course*。
- 通过 Windows 函数 *SetWindowPos()*，可指定屏幕上显示的运行系统窗口的位置和尺寸。在本实例中，将运行系统窗口定位于屏幕的左上角(位置为 0/0)，并且将其大小设为 1024 x 768。
- 上述程序代码中的其余语句执行与本实例不相关的初始化。

4.10.2 实例 2 - 读取系统时间

本实例将说明如何读取并显示系统时间。在本实例中，将显示时间和日期。在画面 *cc_0_startpicture_00.PDL* 中组态本实例。



静态文本时间的 C 动作

```
#include "apdefap.h"
 char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

#pragma code("kernel32.dll")
VOID GetLocalTime(LPSYSTEMTIME lpSystemTime);
#pragma code()

SYSTEMTIME sysTime;
char szTime[6] = "";

GetLocalTime(&sysTime);

sprintf(szTime,"%02d:%02d",sysTime.wHour,sysTime.wMinute);

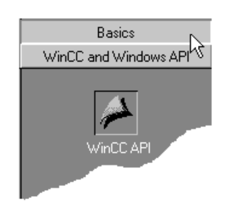
return szTime;

}
```

- 在第一部分中，集成 Windows DLL Kernel32。由于只需要 DLL 的一个函数，因此直接声明该函数。
- 接下来，定义 *SYSTEMTIME* 数据类型的变量 *sysTime*。这是一个结构类型，用于存储系统时间。
- 然后，定义一个字符串变量 *szTime*，用于接受 hh:mm 格式的当前时间。
- 通过 Windows 函数 *GetLocalTime()*，将当前的系统时间写入变量 *sysTime* 中。
- 最后，将当前的系统时间设置为 hh:mm 格式，并通过 sprintf()函数将其作为返回值返回。在*属性* → *其它* → *工具提示文本处*，按照所述步骤创建另一个 *C 动作*。该 C 动作传递当前的日期。
- 函数的执行周期为 *1s*。

4.10.3 实例 3 - 播放声音文件

本实例将说明如何播放声音文件。在本实例中，如果从浏览栏*基础*切换到浏览栏 *WinCC* 和 *Windows API*，则将播放一个声音文件，反之亦然。该实例在画面 *cc_2_keyboard_01.PDL* 中为对象*按钮 1* 组态。



项目函数 CC_PlaySound()

```
#include "apdefap.h"

void CC_PlaySound(char* lpszSoundFile)
{

#pragma code("winmm.dll")
BOOL PlaySound(LPCTSTR lpszSound,HMODULE hModule, DWORD dwSound);
#define SND_FILENAME 0x00020000L
#define SND_ASYNC 0x0001
#pragma code()


BOOL bRet = FALSE;
char szProjectPath[_MAX_PATH];
char szSoundPath[_MAX_PATH];

GetProjectPath(szProjectPath);

sprintf(szSoundPath,"%sSound\\%s",szProjectPath,lpszSoundFile);

bRet = PlaySound(szSoundPath,NULL,SND_FILENAME|SND_ASYNC);

if (bRet == FALSE)
{
      MessageBeep((WORD)-1);
}

}
```

- 在第一部分中，集成 *apdefap.h* 文件。通过该文件，当前项目函数也可以调用其它项目函数。
- 函数标题定义了一个字符串变量，作为传送参数。使用该变量，可传送要播放的声音文件的名称。
- 在第二部分中，集成 Windows DLL *winmm*。由于只需要 DLL 的一个函数，因此直接声明该函数。此外，还定义两个符号常量。
- 该*项目函数*假定项目文件夹中存在一个声音子文件夹。在该文件夹中，存储项目中使用的声音文件。所期望声音文件的路径包括项目路径、声音文件夹的名称以及所传送的声音文件的名称。它将存储在变量 *szSoundPath* 中。
- 通过 Windows 函数 *PlaySound()*即可播放该声音。如果不能播放声音文件，则通过 Windows 函数 *MessageBeep()*产生简短的蜂鸣声来代替声音文件。

4.10.4 实例 4 - 启动程序

本实例说明了启动程序的过程。为此，可利用一个使用 Windows API 的现有*标准函数*。在 *cc_0_startpicture_00.PDL* 画面中组态本实例。



标准函数 ProgramExecute()

```
unsigned int ProgramExecute( char* Program_Name )
{
      // This function will start any Windows Programm
      // if return value > 31 the programm started successfully

      return (  WinExec( Program_Name,
                   SW_SHOWNORMAL )   );
}
```

- *标准函数 ProgramExecute()*只是将所传送的参数转发给 Windows 函数 *WinExec()*。函数 *WinExec()*的返回值将转发给函数 *ProgramExecute()*的调用者。如果返回值大于 31，则程序启动成功。

图形对象执行的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

      ProgramExecute("calc.exe");

}
```

- 通过*标准函数 ProgramExecute()*启动程序 *calc.exe*。它是 Windows 计算器程序。不需要指定任何路径，因为对于 Windows 文件夹中的程序来说没有这个必要。

4.11 标准对话框

在 WinCC 项目 Project_C_Course 中，有关标准对话框主题的实例可以通过单击如下所示的浏览栏图标来访问。实例在画面 cc_9_example_12.PDL 中组态。



Standard Dialogboxes

常规信息

在 WinCC 中创建对话框的一般过程包括创建一个 WinCC 画面以及用画面窗口显示该画面。也可以用 C 动作或其它函数来创建标准对话框。在这种情况下，WinCC 标准对话框以及 Windows 对话框均可使用。
在本章中，对某些可用的标准对话框的应用进行了说明。还有许多可用的标准对话框在此处没有提及。有关这些对话框的信息可参见 WinCC ODK 和 Windows API 文档。

4.11.1 实例 1 - 语言切换

本实例将说明如何使用进行语言切换的 WinCC 标准对话框。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 1* 组态了本实例。

Language Switch

按钮 1 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

HWND hwndParent = NULL;
DWORD dwFlags = 0;
DWORD dwSetLocaleIDs[3] = { 0x0409,0x0407,0x040C };
UINT  uSetIDArraySize = 3;
DWORD dwGetLocaleID;
BOOL bRet;
CMN_ERROR Error;

hwndParent = FindWindow(NULL,"WinCC C-Course");

//set cs language (dwGetLacaleID contains selected language ID)
bRet = DMShowLanguageDialog(hwndParent,dwFlags,dwSetLocaleIDs,
                            uSetIDArraySize,&dwGetLocaleID,&Error);

if (bRet == FALSE)
{
      printf("\r\nError in DMShowLanguageDialog()\r\n"
             "\t%s\r\n",Error.szErrorText);
      return;
}

//set rt language
bRet = SetLanguage(dwGetLocaleID);

if (bRet == FALSE)
{
      printf("\r\nError in SetLanguage()\r\n");
      return;
}

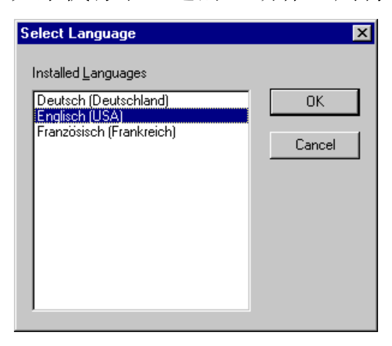
}
```

- 在第一部分中，定义所使用的变量。其中，还定义一个包括三种期望语言的 ID 的向量。
- 通过 Windows 函数 *FindWindow()*，可使用运行系统窗口的窗口标题来确定其窗口句柄。注意：在该代码实例中指定的窗口标题并非运行系统窗口的缺省标题。
- 通过 API 函数 *DMShowLanguageDialog()*，可显示用于进行语言切换的标准对话框。将一个向量传送给此函数，该向量具有要在对话框中显示的语言的 ID。函数将用户所选语言的 ID 写入所传送的变量 *dwGetLocaleID* 中。

- 检查 API 函数 *DMShowLanguageDialog()*的返回值。 其中，如果用户通过单击*取消*终止对话框，则该返回值将具有值 *FALSE*。
- 此处所使用的对话框仅切换 CS 语言。要切换 RT 语言，必须使用*内部函数 SetLanguage()*。将对话框中所选语言的 ID 传送给该函数。

选择语言对话框

如果执行了上述的 *C 动作*，则将显示以下对话框：



4.11.2 实例 2 - 变量选择

本实例将说明如何使用进行变量选择的 WinCC 标准对话框。对话框内所选变量的内容显示在 *I/O 域*中。在*事件 → 鼠标 → 鼠标动作*处为如下所示的对象*按钮 2* 组态了本实例。

Tag Selection

按钮 2 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{
//include file with the LinkType definitions
#include "trigger.h"

BOOL bRet;
char szProjectFile[_MAX_PATH+1];
CMN_ERROR Error;
HWND hwndParent = NULL;
DM_VARKEY dmVarKey;
LINKINFO link;

////////////////////////////////////////////////////////////////////////////
//select tag
if ( DMGetRuntimeProject(szProjectFile,_MAX_PATH+1,&Error) == FALSE)
{
      printf("\r\nError in DMGetRuntimeProject()\r\n"
             "\t%s\r\n",Error.szErrorText);
      return;
}

hwndParent = FindWindow(NULL,"WinCC C-Course");

if ( DMShowVarDatabase(szProjectFile,hwndParent,NULL,NULL,
                       &dmVarKey,&Error) == FALSE)
{
      printf("\r\nError in DMShowVarDatabase()\r\n"
             "\t%s\r\n",Error.szErrorText);
      return;
}

////////////////////////////////////////////////////////////////////////////
//display tag selection
SetText(lpszPictureName,"TagName",dmVarKey.szName);

link.LinkType = BUBRT_LT_VARIABLE_DIRECT;
link.dwCycle = 0;
strcpy(link.szLinkName,dmVarKey.szName);

PDLRTSetLink(0,lpszPictureName,"TagValue","OutputValue",
             &link,NULL,NULL,&Error);

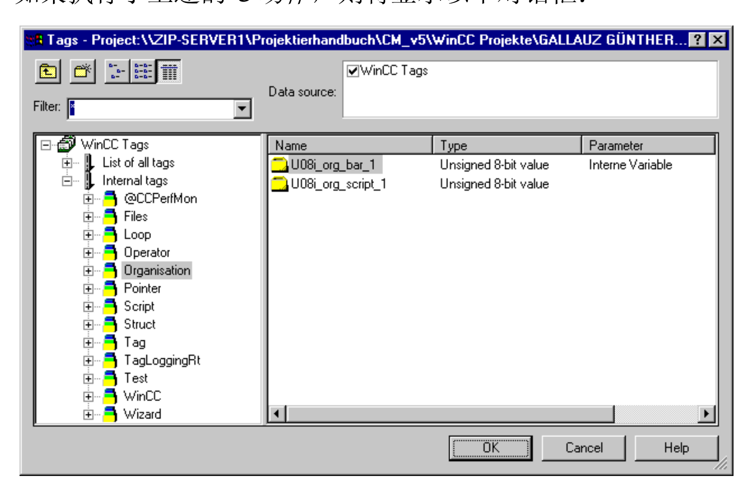
}
```

- 在第一部分中，集成 *trigger.h* 文件。该文件包含本实例中所使用的符号常量的定义。
- 在第二部分中，定义所使用的变量。其中，定义了用于接受有关对话框中所选 WinCC 变量的信息的变量 *dmVarKey*，以及用于接受有关变量连接的信息的变量 *link*。
- 通过 API 函数 DMGetRuntimeProject()确定项目名称。

- 通过 Windows 函数 *FindWindow()*，可使用运行系统窗口的窗口标题来确定其窗口句柄。
- 通过 API 函数 *DMShowVarDatabase()*打开变量选择对话框。有关对话框中所选 WinCC 变量的信息存储在所传送的变量 *dmVarName* 中。
- 如果已经选择了一个变量，则其名称将显示在*静态文本*域中，而其内容将显示在 *I/O 域*中。

选择变量对话框

如果执行了上述的 *C 动作*，则将显示以下对话框：



4.11.3 实例 3 - 出错框

本实例将说明如何才能显示 Windows 出错框。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 3* 组态了本实例。



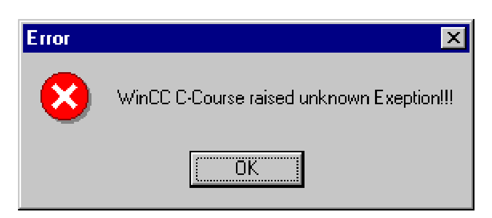
按钮 3 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{
HWND hWnd = NULL;
hWnd = FindWindow(NULL,"WinCC C-Course");
MessageBox(hWnd,"WinCC C-Course raised unknown Exeption!!!",
      "Error",MB_OK|MB_ICONSTOP|MB_APPLMODAL);
}
```

- 在第一部分中，定义 *HWND* 数据类型的变量 *hWnd*。通过 Windows 函数 *FindWindow()*将运行系统窗口的窗口句柄分配给该变量。
- 通过 Windows 函数 *MessageBox()*打开一个出错框。将出错文本指定为第二个参数，出错框的标题指定为第三个参数。第四个参数则指定出错框的外观和状态。出错框只是包含一个*确定*按钮(*MB_OK*)、显示出错符号(*MB_ICONSTOP*)并且处于模态(*MB_APPLMODAL*)。由此可见，用户在往下进行之前，必须首先确认出错框。
- 如果已经选择了一个变量，则其名称将显示在*静态文本*域中，而其内容将显示在 *I/O 域*中。

出错框

如果执行了上述的 *C 动作*，则将显示以下出错框：



4.11.4 实例 4 - 询问框

本实例将说明如何才能显示 Windows 询问框，以及如何根据用户所按下的按钮来执行另一个动作。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 4* 组态了本实例。



按钮 4 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{

HWND hWnd = NULL;
int iRet;

hWnd = FindWindow(NULL,"WinCC C-Course");

iRet = MessageBox(hWnd,"Do you want to do something?","Question",
                        MB_YESNO|MB_ICONQUESTION|MB_APPLMODAL);

printf("\r\nExample 3\r\n");

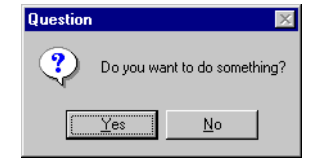
if (iRet == IDYES)
{
      printf("User selected YES button\r\n");
}
else // if (iRet == IDNO)
{
      printf("User selected NO button\r\n");
}

}
```

- 在第一部分中，定义 *HWND* 数据类型的变量 *hWnd*。此外，还定义 int 类型的变量 *iRet*。
- 通过 Windows 函数 *FindWindow()*，可使用运行系统窗口的窗口标题来确定其窗口句柄。
- 通过 Windows 函数 *MessageBox()*打开一个询问框。第四个参数指定询问框的外观和状态。询问框只是包含是与否按钮(*MB_YESNO*)、显示一个问号(*MB_ICONQUESTION*)并且处于模态(*MB_APPLMODAL*)。该函数的返回值存储在变量 *iRet* 中。
- 在最后一部分中，对函数的返回值进行分析。如果用*是*按钮来结束对话框，则返回值为 *IDYES*；如果用否按钮来结束对话框，则返回值为 *IDNO*。

询问框

如果执行了上述的 *C 动作*，则将显示以下询问框：



4.11.5 实例 5 - 打开标准对话框

本实例将说明如何显示用于打开文件的标准对话框。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 5*组态了本实例。

C Data Types

按钮 5 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{

#pragma code ("comdlg32.dll")
#include "commdlg.h"
#pragma code()

BOOL bRet;
OPENFILENAME ofn;
char szFilter[] = "Textfiles|*.txt|All Files|*.*|";
char* psz;
char szFile[_MAX_PATH+1];
char szInitialDir[_MAX_PATH+1] = "C:\\";

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = FindWindow(NULL,"WinCC C-Course");

for (psz = szFilter; *psz; psz++)
{
       if (*psz == '|')
       {
              *psz = 0;
       }
}
ofn.lpstrFilter = szFilter;

ofn.lpstrFile = szFile;
ofn.nMaxFile = _MAX_PATH+1;

GetProjectPath(szInitialDir); //if function fails initial
                              //directory is "C:\"
ofn.lpstrInitialDir = szInitialDir;

bRet = GetOpenFileName(&ofn);

if (bRet == FALSE)
{
       printf("\r\nError in GetOpenFileName()\r\n");
       return;
}

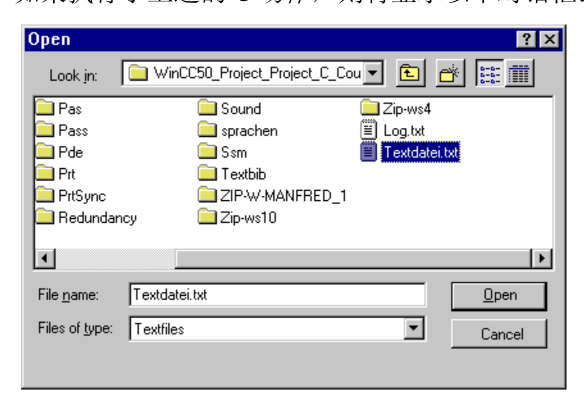
printf("\r\nSelected File (Path+Name)\r\n%s\r\n",ofn.lpstrFile);

}
```

- 在第一部分中，集成 Windows DLL *comdlg32*。
- 在第二部分中，定义所需的变量。其中，定义了 *OPENFILENAME* 结构类型的变量 *ofn*。
- 在第三部分中，将信息加入变量 ofn 中。
- 将变量 ofn 传送给 Windows 函数 *GetOpenFileName()*。该函数打开文件选择对话框。用户所选文件的名称存储在变量 *ofn* 中。输出所选文件的名称。

打开标准对话框

如果执行了上述的 *C 动作*，则将显示以下对话框：



附加的实例

本章中随后的实例将如同实例 5 一样对标准文件对话框进行说明。
在实例 6 中，将使用*另存*为对话框。
在实例 7 中，将创建*项目函数 GetFileName()*，它将使标准文件对话框的处理更容易。根据所传送的符号常量，该函数可以显示*打开*对话框或*另存*为对话框。可用的符号常量有 *GFN_OPEN 和 GFN_SAVE*。

4.12 文件

在 WinCC 项目 Project_C_Course 中，有关文件主题的实例可以通过单击如下所示的浏览栏图标来访问。实例在画面 cc_9_example_13.PDL 中进行组态。



Files

打开文件

在 C 语言中，不管其内容如何，文件总是作为字符集显示。在 C 动作或另一个函数中使用一个文件之前，必须先将文件打开。如果该文件使用结束，则应该将其关闭。
文件用 fopen()函数来打开。在如下所示的程序代码中，显示了 fopen()函数的应用。

```
FILE* pFile = NULL;
pFile = fopen("C:\\Test.txt","r");
```

为了能够使用文件，必须定义一个指向该文件的指针。为此，可使用 FILE*数据类型。如果文件打开失败，则 fopen()函数返回指向所打开文件的指针或 NULL。具有路径说明的要打开文件的名称必须作为第一个参数传送给 fopen()函数。打开文件所使用的模式(例如用于读)作为第二个参数传送给函数。可以为模式指定的值列于下表中。

模式	描述
r	打开文件进行读。如果文件不存在或没有读取权限，则返回值为 *NULL*。
w	打开文件进行写。如果文件不存在或没有写入权限，则返回值为 NULL。
a	打开文件，向文件末尾添加数据。如果文件不存在，则创建文件。如果不能创建文件或不能写入文件，则返回值为 *NULL*。
r+	打开文件进行读和写。如果文件不存在或没有该文件的读取与写入权限，则返回值为 *NULL*。
w+	创建一个文件进行读和写。如果文件已经存在，则将删除该文件。如果没有进行这些动作的权限，则返回值为 *NULL*。
a+	打开文件，进行读或向文件末尾添加数据。如果文件不存在，则创建文件。如果没有该文件的读取和写入权限，则返回值为 NULL。

关闭文件

在文件使用结束之后，应该将其关闭。文件用 fclose()函数来关闭。在如下所示的程序代码中，显示了 fclose()函数的应用。将指向要关闭的文件的指针传送给该函数。

```
fclose(pFile);
```

写文件和读文件

要写文件，可使用与 printf()函数相类似的函数。它就是 fprintf()函数。fprintf()函数的应用与 printf()函数的应用遵循同一原理。但是，它输出到文件，而不是输出到全局脚本诊断窗口。函数将指向该文件的指针作为第一个参数。在如下所示的程序代码中，显示了 fprintf()函数的应用。

```
fprintf(pFile,"%d\r\n%f\r\n",iValue,dValue);
```

要读文件，可使用 fscanf()函数。fscanf()函数在结构上与 fprintf()函数完全相同。然而，它不是指定其值要写入文件的变量，而是指定要将文件内容写入其中的变量的地址。

4.12.1 实例 1 - 保护数据

本实例将说明如何将数据写入文件中。首先从 WinCC 变量中读取要写入的数据。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 1* 组态了本实例。

Back-Up Data

按钮 1 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{

FILE* pFile = NULL;

char szFile[_MAX_PATH+10];

int iData;
float fData;

//get project path
if (GetProjectPath(szFile) == FALSE)
{
      printf("\r\nError in GetProjectPath()\r\n");
      return;
}

//create file name
strcat(szFile,"Data.txt");

//open or create file to write
pFile = fopen(szFile,"w+");

//check return value of fopen()
if (pFile == NULL)
{
      printf("\r\nError in fopen()\r\n");
      return;
}

//get data to write
iData    = GetTagSDWord("S32i_course_file_1");
fData    = GetTagFloat("F32i_course_file_1");

//write data
fprintf(pFile,"%d\r\n%f\r\n",iData,fData);

fclose(pFile);

//output in diagnostics window
printf("\r\nData written in file:\r\n\t%d\r\n\t%f\r\n",
            iData,fData);

}
```

- 在第一部分中，定义所需的变量。其中，定义并初始化了一个 *FILE**类型的变量。
- 通过*项目函数 GetProjectPath()*确定项目路径。
- 下一步，通过 *strcat()*函数编辑所要创建的文件的路径。将该路径传送给 *fopen()*函数。通过该函数，可打开或创建所期望的文件。
- 在下一部分中，从 WinCC 变量内读取所要写入的数据。
- 通过 *fprintf()*函数，将数据写入文件中。然后，再将文件关闭。

4.12.2 实例 2 - 读取数据

本实例将说明如何从文件中读取数据。将读取的数据写入 WinCC 变量中。在*事件 → 鼠标 → 鼠标动作*处为如下所示的对象*按钮 2* 组态了本实例。

Read Data

按钮 2 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

FILE* pFile = NULL;

char szFile[_MAX_PATH+10];

int iData;
float fData;

//get project path
if (GetProjectPath(szFile) == FALSE)
{
	printf("\r\nError in GetProjectPath()\r\n");
	return;
}

//create file name
strcat(szFile,"Data.txt");

//open file to read
pFile = fopen(szFile,"r+");

//check return value of fopen()
if (pFile == NULL)
{
	printf("\r\nError in fopen()\r\n");
	return;
}

//read data
fscanf(pFile,"%d\r\n%f\r\n",&iData,&fData);

fclose(pFile);

//set data
SetTagSDWord("S32i_course_file_1",iData);
SetTagFloat("F32i_course_file_1",fData);

//output in diagnostics window
printf("\r\nData read from file:\r\n\t%d\r\n\t%f\r\n",
		iData,fData);

}
```

- 在第一部分中，定义所需的变量。其中，定义并初始化了一个 *FILE** 类型的变量。
- 通过*项目函数 GetProjectPath()*确定项目路径。
- 下一步，通过 *strcat()*函数编辑所要打开的文件的路径。将该路径传送给 *fopen()*函数。使用该函数，可打开要读取的期望文件。
- 通过 *fscanf()*函数，从文件中读取数据。然后，再将文件关闭。
- 在下一部分中，将读取的数据写入 WinCC 变量中。

4.12.3 实例 3 - 报表

本实例将说明如何创建报表文件。创建一个*项目函数*，并将报表文本传送给它。然后将该文本写入报表文件中。在*事件* → *鼠标* → *鼠标动作*处为如下所示的对象*按钮 3* 组态了本实例。

Report

项目函数 LogText()

```
#include "apdefap.h"

BOOL LogText(char* lpszLogText)
{

FILE* pFile = NULL;

char szFile[_MAX_PATH+10];

//get project path
if (GetProjectPath(szFile) == FALSE)
{
      printf("\r\nError in GetProjectPath()\r\n");
      return FALSE;
}

//create file name
strcat(szFile,"Log.txt");

//open or create file to append
pFile = fopen(szFile,"a+");

//check return value of fopen()
if (pFile == NULL)
{
      printf("\r\nError in fopen()\r\n");
      return FALSE;
}

//append data
fprintf(pFile,"%s - %s\r\n",GetLocalTimeString(),lpszLogText)

fclose(pFile);

return TRUE;

}
```

- 为函数分配一个字符串变量，它将添加至报表文件的末尾。
- 在第一部分，定义所需的变量。其中，定义并初始化了一个 *FILE** 类型的变量。
- 通过项目函数 *GetProjectPath()* 确定项目路径。
- 下一步，通过 *strcat()* 函数编辑所要打开的文件的路径。将该路径传送给 *fopen()* 函数。使用该函数，可打开要添加数据的期望文件。
- 通过 *fprintf()* 函数，将所传送的报表文本输入到文件中。在输入各个报表条目之前，通过*标准函数 GetLocalTimeString()* 输入当前的系统时间。然后，再将文件关闭。

按钮 3 的 C 动作

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

if (LogText(GetTagChar("T08i_course_file_1")) == FALSE)
{
	printf("\r\nError in LogText()\r\n");
}

}
```

- 在 *C 动作*中，读取 WinCC 文本变量的内容，并将其传送给先前所创建的*项目函数 LogText()*。从而，WinCC 文本变量的内容就被输入到报表文件中。

4.13 动态向导

在 WinCC 项目 Project_C_Course 中，有关动态向导主题的实例可以通过单击如下所示的浏览栏图标来访问。实例在画面 cc_9_example_14.PDL 中进行组态。

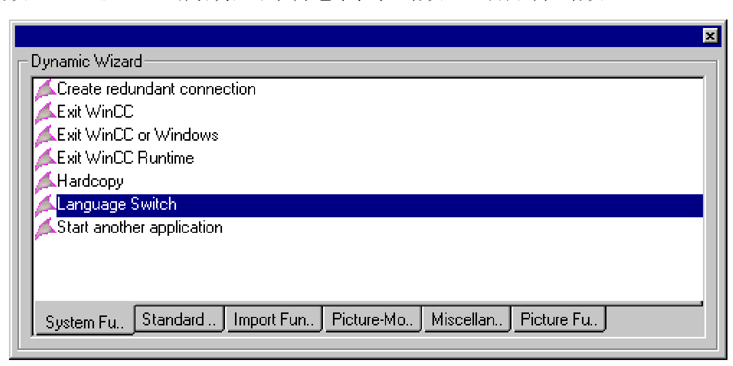

Dynamic Wizard

常规信息

动态向导可作为附加函数在图形编辑器中使用。它可以支持用户经常重复的组态过程。这将简化组态工作，并且可以减少可能发生的组态错误。
动态向导由各种不同的动态向导函数组成。已经有许多可用的动态向导函数。它们可以由用户定义的函数进行补充。

使用动态向导

动态向导通过查看 → 工具栏 → 动态向导菜单显示在图形编辑器中。下面显示了动态向导的结构。在各种不同的标签中，显示了已经可以使用的动态向导函数。通过 D 所期望的动态向导函数，可启动函数。



一个动态向导函数由多个页面组成，这些页面必须由用户进行填写。这些页面包括起始页面、触发页面、不同选项页面和最终页面，该最终页面对前面页面中所进行的设置进行总结。

4.13.1 动态向导函数的创建

要创建*用户定义的动态向导*函数，可使用一个独立的编辑器。该编辑器位于*bin*文件夹中：即 *dynwizedit.exe* 程序。
*每个动态向导*函数都存储在单独的脚本文件中。对于德语、英语和法语等各种语言来说，均存在一个单独的脚本文件。根据语言的不同，这些脚本文件存储在下列文件夹中：
WinCC 安装文件夹\Wscripts\Wscripts.deu
WinCC 安装文件夹\Wscripts\Wscripts.enu
WinCC 安装文件夹\Wscripts\Wscripts.fra
启动*动态向导编辑器*之后，从工具栏上为创建新*动态向导*函数选择所期望的语言。
*动态向导*函数必须遵循一种规定的结构。在本手册范围内，提供了两个创建*动态向导*函数的实例。这些实例所处的脚本文件存储在特别为此而创建的 *DynWiz* 子文件夹中，该子文件夹属于 WinCC 项目 *Project_C_Course*。必须将这些脚本文件复制到上面列出的文件夹中，即缺省脚本文件所处的文件夹中。然后，可以从*动态向导编辑器*中打开这些实例。

演示向导

在 *Demo.wnf* 脚本文件中，已创建了一个名为*演示向导*的*动态向导*。该向导显示的基本功能可使用户方便地输入数据。然而，此*动态向导*不执行任何动作。

使电机动态化向导

在 *Motor.wnf* 脚本文件中，已经创建了一个名为*使电机动态化的动态向导*。此向导专门为使名为*电机*的*自定义对象*动态化而创建，不能用于任何其它对象类型。该*自定义对象*存储在 WinCC 项目 *Project_C_Course* 的项目库中，并且可以从项目库插入到画面中。*自定义对象电机*通过*动态向导使电机动态化*与*电机*结构类型的 WinCC 结构变量连接。更准确地说就是为该对象创建了各种不同的 *C 动作*和变量连接。假设存在一个内部 WinCC 文本变量 *T08i_course_wiz_selected*。
通过该变量，可标记当前所选的电机对象。

编译脚本文件

已完成创建的*动态向导*函数必须通过*动态向导* → *编译脚本*菜单来进行编译，然后再进行保存。为了在*图形编辑器*中使用*动态向导*函数，必须将其集成到*动态向导*的数据库中。这通过以下菜单来完成，即*动态向导* → *读取向导脚本*。要读取的脚本文件必须从显示的对话框中选择。

4.13.2 动态向导函数的结构

下面将说明组成*动态向导*函数的各个部分。

头文件与 DLL 的集成

*动态向导*函数的第一部分集成所需的头文件。要集成的最重要文件是 *dynamic.h* 文件，在其中声明了与*动态向导*用户界面的外观相关的函数。此外，在此处可以集成所有期望的 Windows 或 WinCC API 的 DLL。

```
#include "dynamic.h"

#pragma code("pdlcsapi.dll")
#include "pdlcsapi.h"
#pragma code()
```

与语言相关的定义

如果*动态向导*函数要在多种语言下都可以使用，则必须为每种语言创建一个单独的脚本文件。因此，与语言相关的文本应该在程序代码的前面进行定义。然后，只需简单地复制为某种语言所创建的脚本文件。接着只要修改与语言相关的定义的部分。

```
/////////////////////////////////////////////////////////////////////////////
//change only this strings to generate wizard scripts for other languages

#include  "defenu.h"

char* DynWizGroupName     = "WinCC C-Course";
char* DynWizDynamicName   = "Make a Motor Dynamic";
char* DynWizToDoOption1   = "Select the desired Structure Tag:";
char* DynWizGenerateInfo = "The motor is made dynamic using the\r\n"
                           "structure tag\r\n%s\r\n.";

//
/////////////////////////////////////////////////////////////////////////////
```

属性列表

可以选择指定是否*动态向导*函数只能用于某几种对象类型。这通过指定对象属性的列表来完成。如果一个对象具有一个或多个已列出的属性，则*动态向导*函数可用于该对象。*动态向导使电机动态化*已经采用该选项来使此向导函数只能用于电机类型的*自定义对象*。该对象类型只具有属性*手动与选择*。如果使用一个空的属性列表，则*动态向导*函数可用于所有的对象类型。无论如何，必须有一个属性列表，即使它是空的。

处理函数

处理函数就是在按下完成按钮之后执行*动态向导*函数的实际工作的函数。该函数的名称必须在系统接口中指定。更广义的处理函数出现在*动态向导使电机动态化*中。

信息函数

信息函数概括了用户所作的设置，并以概要的形式将其显示在*动态向导*函数的最后一页上。该函数的名称必须在系统接口中指定。

```
//this wizard can only be executed on the customized object "motor"
BEGIN_PROPERTY_SCHEME
{ "Hand", VT_BOOL },
{ "Selection", VT_BOOL },
END_PROPERTY_SCHEME
```

系统接口

通过系统接口，指定新*动态向导*函数的各种不同属性。下列代码实例解释了各参数的含义。

```
BEGIN_DYNAMICS
      {
            DynWizGroupName,          //group name
            DynWizDynamicName,        //dynamic name
            NULL,
            "logo16.bmp",             //use the default icon
            NULL,                     //no help string is used
            {
                  "OnOption1",
                  "OnOption2",
                  NULL,
            },
            "OnGenerate",
            "OnShowGenerateInfo",
            {
                  JCR_TRIGGERS,
                  { NULL, NULL },
            },
      },
END_DYNAMICS
```

- 第一个参数指定*动态向导*函数显示在哪个标签中。
- 第二个参数指定*动态向导*函数在哪个名称下显示。
- 至于第三个参数，则总是传送 *NULL*。
- 第四个参数包含用于*动态向导*函数的图标名称。
- 作为第五个参数进行传送的是帮助文本，它更详细地描述了*动态向导*函数的功能。

  第六个参数指定一个列表，它包含所创建函数的各选项页的名称。该列表必须用 *NULL* 条目来结束。最多可创建五个选项页面。

- 第七个参数指定处理函数的名称，该函数在按下*完成*按钮之后进行调用。
- 第八个参数指定函数的名称，该函数概括了选项页面中所作的设置，并在用户按下*完成*按钮之前显示它们。
- 第九个参数指定的是要在触发页面上显示的触发器列表。对于更经常发生的应用情况，可利用宏来填写该触发器列表。

全局变量

对于要在选项页中设置的每个参数，必须定义一个全局变量。这将保证所设置的参数对所有已创建的函数均是已知的并且可以使用。

```
//this wizard can only be executed on the customized object "motor"
BEGIN_PROPERTY_SCHEME
{ "Hand", VT_BOOL },
{ "Selection", VT_BOOL },
END_PROPERTY_SCHEME
```

选项页面

对于所需的每个选项页面，必须创建一个单独的函数。这些函数的名称必须在系统接口中指定。

5 附录

附录包含了未直接放入*组态手册*中的主题集。

5.1 提示和诀窍

用 WinCC 进行组态的附加实例。

5.1.1 在 I/O 域处的格式化的输入/输出

为了使 I/O 域能显示格式化的数值或者将格式化的输入值传送给 PLC，必须组态下列动作：
在 I/O 域的输出值属性处的动作(重要：如果需要有小数位，则用浮点数型)：

```
Float a;
a=GetTagFloat("DB21_DW1");
return(a/100);
```

在 I/O 域的输入值事件处的动作(变量 Var1 是一个无符号的 16 位数)：

```
float a;
a=GetInputValueDouble(lpszPictureName,lpszObjectName);
SetTagFloat("Var1",a*100);
```

5.1.2 打开画面处指定对象的动作

在打开画面时，存在画面中一个或多个对象的属性处的动作只执行一次的情况。一种选择是在*事件* → *其它* → *打开画面*处将画面对象的指定画面的动作公式化。然而这也存在缺点，即动作必须作用于画面中的对象，因此必须在动作中明确指定对象名称。对象不能够再进行自由处理。此解决方法不是面向对象的。
有一个选项可以避免出现这个难题：

- 定义一个内部变量(例如 *dummy*)，它从不更新或特意设置。将对象处动作的触发设置为一旦该变量改变。在运行系统中打开画面时，动作被激活一次，然后只有当变量 *dummy* 改变时，动作才会再次响应，但因为变量根本不会改变，所以不会发生这种情况。

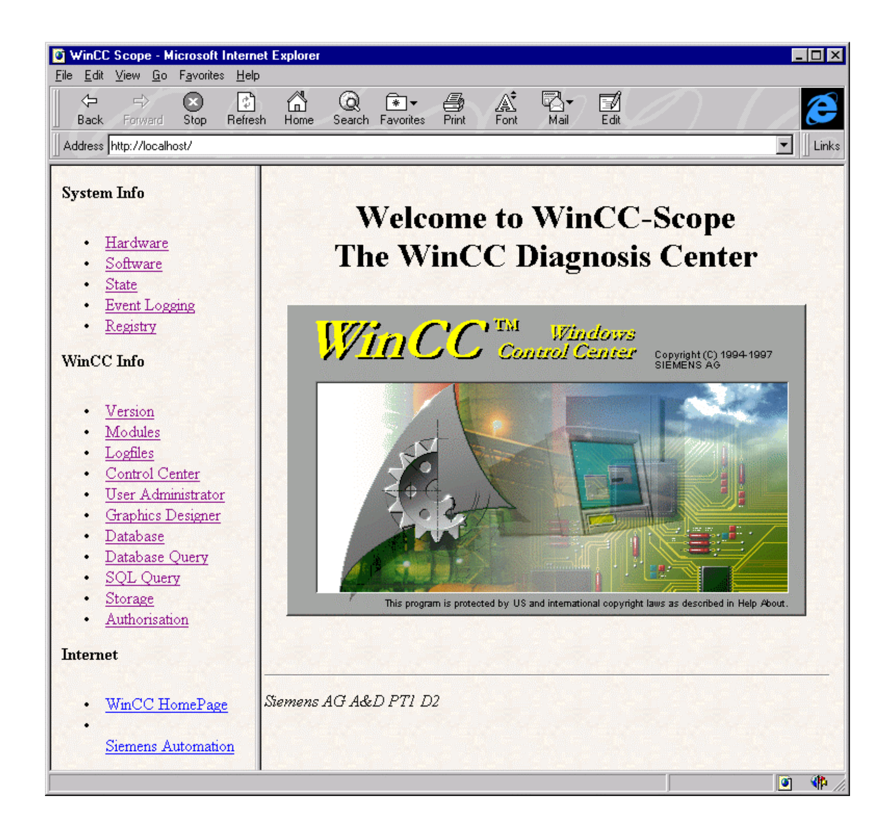
5.1.3 WinCC Scope

常规信息

WinCC *Scope* 是支持诊断 WinCC 项目的工具。它提供了有关激活的项目和特定计算机系统的信息。要使用 Scope，需要有个 Web 浏览器(例如 Internet Explorer)。此外，还必须安装 TCP/IP 网络协议。

启动和运行

如果已经安装 WinCC，那么 Scope 也被缺省安装。在使用 Scope 之前，必须启动 *WinCCDiagAgent.exe* 程序。它位于 *Siemens\WinCC\WinCCScope\bin* 文件夹内。该程序是个简单的 HTTP 服务器。在此之后，可从开始菜单中启动 *Scope*。在起始页中，有关 *WinCC Scope* 运行的常规描述可以通过*如何使用新的诊断接口*链接来访问。单击链接 http://localhost 来启动 *Scope*。根据左边的列表，可以获得各种不同的信息。在系统信息这一部分中，可以访问有关计算机系统的常规信息，在 *WinCC 信息*这一部分中，可以访问有关当前激活的 WinCC 项目的信息。



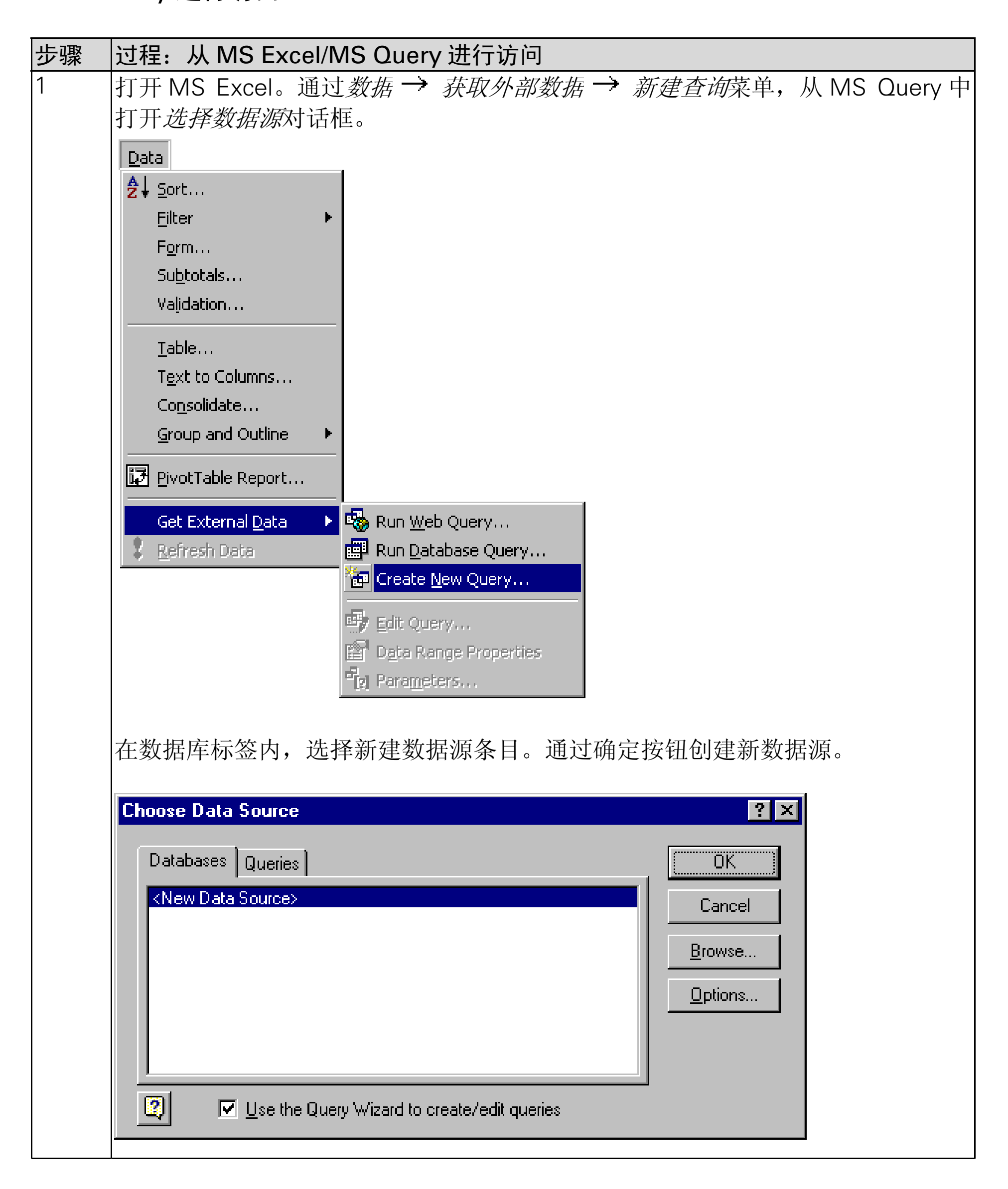
5.1.4 访问数据库

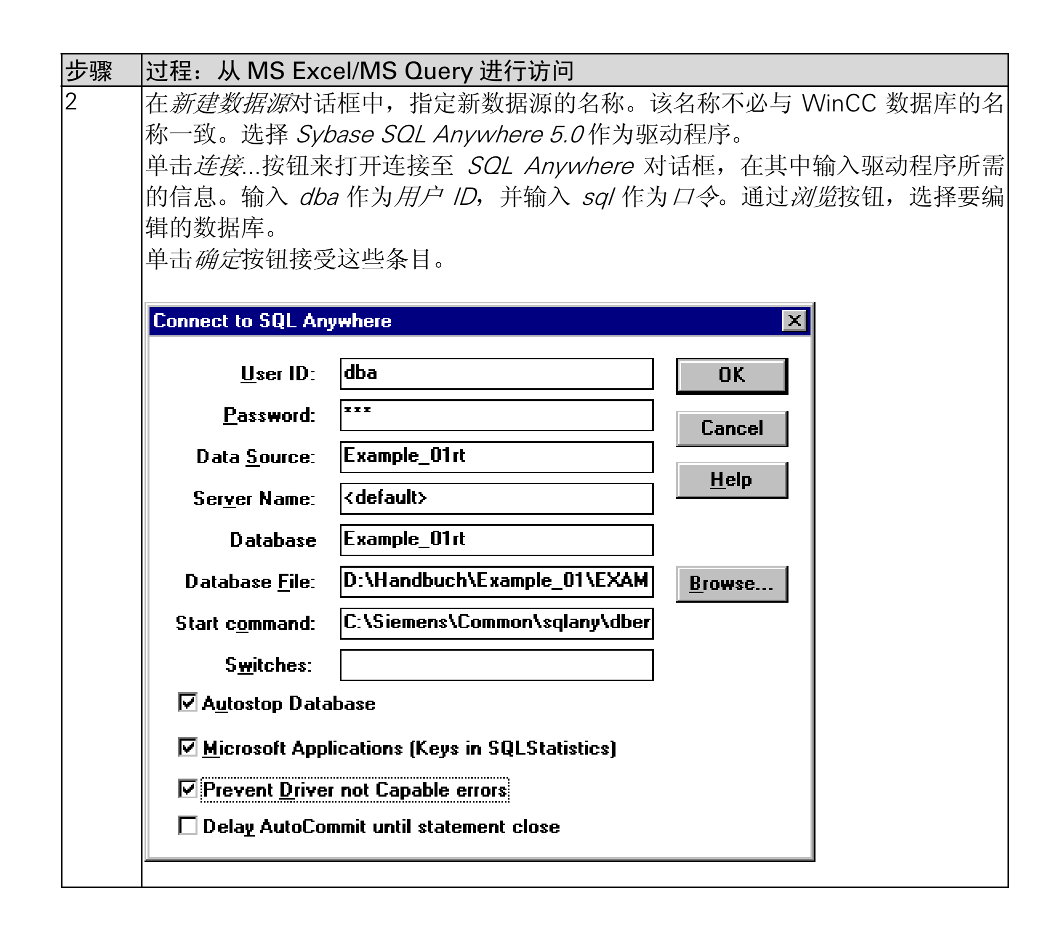
5.1.4.1 从 MS Excel/MS Query 访问数据库

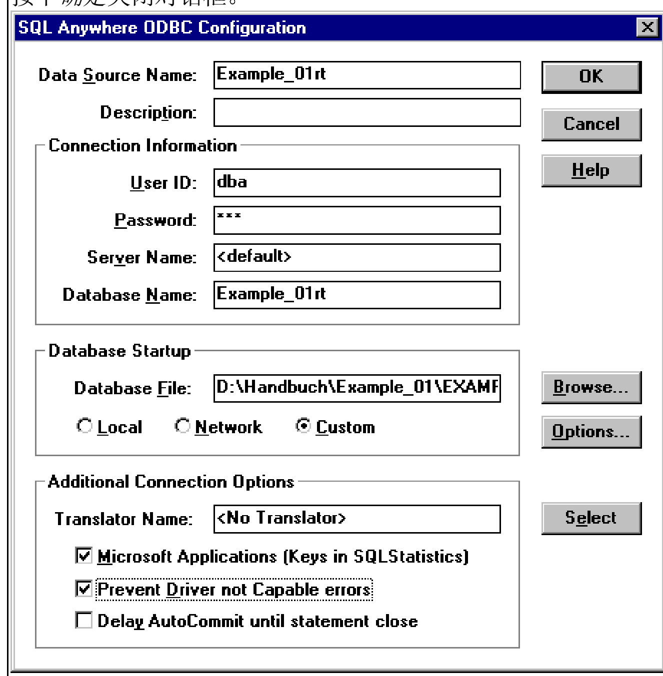
以下有关对 WinCC 数据库进行访问的描述指的是应用带 SR-1 的 Microsoft® Excel 97。

从 MS Excel/MS Query 进行访问

步骤	过程：从 MS Excel/MS Query 进行访问
1	打开 MS Excel。通过*数据* → *获取外部数据* → *新建查询*菜单，从 MS Query 中打开*选择数据源*对话框。 在数据库标签内，选择新建数据源条目。通过确定按钮创建新数据源。



步骤	过程：从 MS Excel/MS Query 进行访问
2	在*新建数据源*对话框中，指定新数据源的名称。该名称不必与 WinCC 数据库的名称一致。选择 *Sybase SQL Anywhere 5.0* 作为驱动程序。 单击*连接...*按钮来打开连接至 *SQL Anywhere* 对话框，在其中输入驱动程序所需的信息。输入 *dba* 作为*用户 ID*，并输入 *sql* 作为*口令*。通过*浏览*按钮，选择要编辑的数据库。 单击*确定*按钮接受这些条目。 

步骤	过程：从 MS Excel/MS Query 进行访问
3	如果还没有为所选择的数据库组态数据源，则显示消息*未找到数据源名称以及未指定缺省驱动程序*。 确认此消息，并再次按下*连接*...按钮。在*选择数据源*对话框中，选择*机器数据源*标签。从数据源列表中已获得 CS 以及当前运行的 WinCC 项目的运行系统数据库。这些数据源的名称以 CC_开头，后跟项目名称。表示运行系统数据库的数据源名称以字符 R 结尾。 但是，如果要编辑任意一个 WinCC 数据库，则首先必须为其创建数据源。这通过*新建*按钮来完成。在所显示的*新建数据源*向导的第一页上，选择用户数据源条目。通过单击*下一步*结束这一页。在下一页上，选择驱动程序 *Sybase SQL Anywhere 5.0*。通过单击下一步*结束*这一页。通过单击完成结束向导的最后一页。 打开 *SQL Anywhere ODBC* 配置对话框，在其中输入驱动程序所需的信息。输入 *dba* 作为*用户 ID*，并输入 *sql* 作为*口令*。通过*浏览*按钮，选择要编辑的数据库。 按下*确定*关闭对话框。  在*选择数据源*对话框中选择新创建的数据源，然后用*确定*按钮关闭该对话框。 确认连接至 *SQL Anywhere* 对话框。 数据源的组态也可以先通过 Windows 控制面板来完成。在 Windows 控制面板中，打开 *ODBC 数据源管理器*。单击*添加*按钮同样调用*新建数据源*向导。 

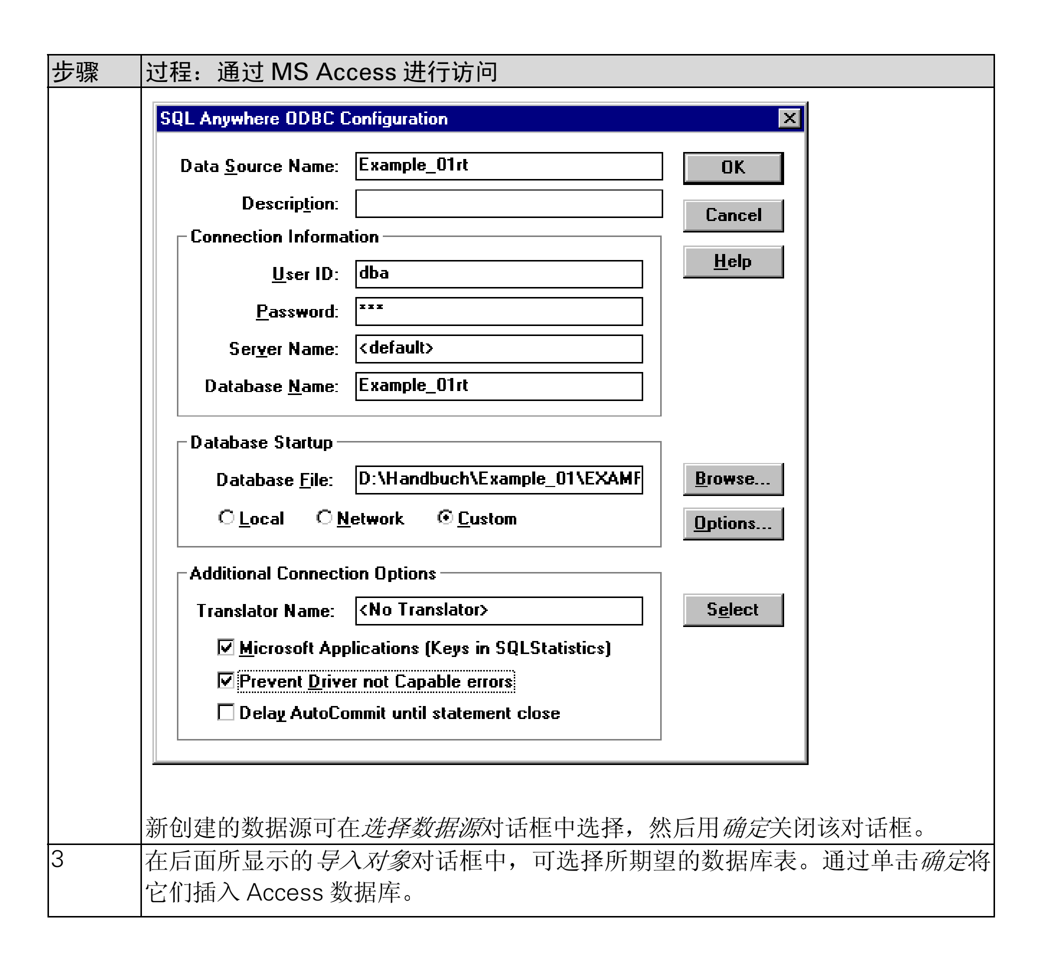
步骤	过程：从 MS Excel/MS Query 进行访问
4	通过确定按钮关闭*新建数据源*对话框。 在*选择数据源*对话框中选择新创建的数据源，然后用*确定*按钮关闭该对话框。 在所显示的*查询向导*的第一页上，列出了所有可用的表和列。选择期望的表和列，然后通过单击*下一步*关闭该页。在下面几页中可以设置数据的过滤器及其排序顺序。在最后一页中，指定数据是否要在 MS Excel 或 MS Query 中做进一步处理。单击*完成*关闭对话框。
5	在所显示的将*外部数据返回 Microsoft Excel* 对话框中，指定要插入表的位置。此外，还可以指定外部数据范围的属性。单击*确定*按钮关闭对话框。

5.1.4.2 从 MS Access 访问数据库

以下有关对 WinCC 数据库进行访问的描述指的是应用带 SR-1 的 Microsoft® Access 97。

通过 MS Access 进行访问

步骤	过程：通过 MS Access 进行访问
1	打开一个 Access 数据库或新建一个数据库。通过*文件* → *获取外部数据* → *导入...*菜单，打开*导入*对话框。选择列表条目 *ODBC Databases()*作为*文件类型*。 将自动打开*选择数据源*对话框。在机器数据源标签中，选择数据源。从数据源列表中已获得 CS 和当前运行的 WinCC 项目的运行系统数据库。这些数据源的名称以 *CC_*开头，后跟项目名称。表示运行系统数据库的数据源名称以字符 *R* 结尾。
2	如果所期望的 WinCC 数据库不包含在列表中，则必须通过单击*新建*按钮先创建一个新的数据源。 在所显示的*新建数据源*向导的第一页上，选择*用户数据源*条目。通过单击下一页结束该页。在下一页，选择驱动程序 *Sybase SQL Anywhere 5.0*。通过单击 *下一页*结束该页。通过单击完成结束向导的最后一页。 打开 *SQL Anywhere ODBC* 配置对话框，输入驱动程序需要的信息。输入 *dba* 作为*用户 ID*，并输入 *sql* 作为*口令*。通过*浏览*按钮，选择要编辑的数据库。 按下*确定*关闭对话框。

<table>
<tr><th>步骤</th><th>过程：通过 MS Access 进行访问</th></tr>
<tr><td></td><td>


新创建的数据源可在*选择数据源*对话框中选择，然后用*确定*关闭该对话框。</td></tr>
<tr><td>3</td><td>在后面所显示的导入对象对话框中，可选择所期望的数据库表。通过单击确定将它们插入 Access 数据库。</td></tr>
</table>

5.1.4.3 从 ISQL 访问数据库

可以使用 ISQL 直接访问 WinCC 数据库。然而，这完全根据个人情况而定，因为在编辑或删除表格后组态数据可能变得不一致。

通过 ISQL 访问

步骤	过程：通过 ISQL 访问
1	从 Siemens\Common\sqlany 文件夹中启动 ISQL.EXE。 显示*交互式 SQL* 登录对话框。输入 *dba* 作为*用户 ID*，并输入 *sql* 作为*口令*。如果用确定按钮予以确认，则程序将自动与当前打开的 WinCC 数据库(即 CS 数据库)相连。但是，如果要访问另一个数据库，例如运行系统数据库，则通过*命令 →连接菜单*来完成。在下面显示的对话框中，为*用户 ID* 和*口令*输入相同的条目。为*数据库文件*指定包括了完整路径的所期望的数据库。
2	在命令窗口中，现在可以输入 SQL 语句，通过单击*执行*按钮来执行。 以下是一些 SQL 语句的实例： • select* from systable：显示全部表格名称 • select * from >：显示名称为>的表格中的内容 • unload table > to >：将名称为>的表格导出到名称为>的文件中 • drop table >：删除名称为>的表格

5.1.4.4 从 WinCC Scope 访问数据库

通过 WinCC Scope 进行访问

步骤	过程：通过 WinCC Scope 进行访问
1	从 Windows 开始菜单中启动 WinCC Scope 前，必须先激活位于 *Siemens\WinCC\WinCCScope\bin* 文件夹中的应用程序 WinCCDiagAgent.exe。 Scope
2	在第一页中，有关 WinCC Scope 运行的常规描述可以通过*如何使用新的诊断接口*链接来访问。 单击链接 http://localhost 来启动 Scope。
3	在左边，可以从列表中选择各种不同的功能。 • 通过*数据库*条目，显示有关 WinCC 数据库的常规信息。 • 通过*数据库查询*条目，可以显示数据库的各个表格。将当前打开的 WinCC 项目的 CS 数据库预置为*数据源*。该数据源的名称以 *CC_*开头，后跟项目名称。表示运行系统数据库的数据源名称以字符 *R* 结尾。其它数据源也可显示。 • 通过 *SQL 查询*条目，可以将 SQL 语句应用于所选择的数据源。然而，仅当掌握了有关系统的广泛知识时，才能用 SQL 语句来编辑 WinCC 数据库。SQL 语句的实例可参考前面的从 *ISQL 访问数据库*一节。

5.1.4.5 通过 C 动作从数据库导出

也可从 WinCC 运行系统画面激活数据的导出。为此，可以通过 ProgramExcecute 命令行来启动交互式 SQL。要执行的动作存储在命令文件中(在本实例中：archive.sql)。

C 动作，例如一个按钮

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{

char* path =  "C:\\SIEMENS\\Common\\SQLANY\\ISQL -q -b -c";
char* parameters =
          "UID=DBA;PWD=SQL;DBN=CC_Project_97-10-21_09:53:27R";
char* action = "read D:\\WinCC\\Project\\archiv.sql";

char ExportArchive[200];

sprintf(ExportArchive,"%s %s %s",path,parameters,action);

ProgramExecute(ExportArchive);
}
```

- 变量 *path* 包含了至 ISQL.exe 程序的路径及其调用参数。
- 变量 *parameters* 包含了在*交互式 SQL 登录*对话框中进行数据库连接的条目。它们是：
- UID (用户 ID)：DBA
- PWD (口令)：SQL
- DBN (数据库名称)：ODBC 数据源的名称。该数据源的名称以 *CC_*开头，后跟项目名称和项目创建的日期/时间。表示运行系统数据库的数据源名称以字符 R 结尾。当项目激活时，该名称可从 Windows *控制面板 → ODBC → 用户 DSN 标签*中确定。
- 变量 *action* 表示将要执行的在 *archive.sql* 文件中列出的 SQL 语句。
- 这些语句概括在 *ExportArchives* 中，并用 *ProgramExecute()*函数来执行。

注意：
如果要从两个项目数据库之外的数据库导出，则应指定 DBF 参数(包括数据库路径的数据库文件)而不是 DBN 参数。但是，此方法不适用于当前激活的项目数据库。

文件内容：archive.sql

```
select * from PDE#HD#ProcessValueArchive#Analog;
output to D:\WinCC\Projekt\archiv.txt format ascii
```

在打开的数据库内，选择测量值归档 *pde#hd#ProcessValueArchive#Analog*，然后用输出命令导出到 ASCII 文件 archiv.txt。

5.1.4.6 数据库选择

前面描述的命令文件中的 *select* 命令用于选择表。这些表的子集可以连同附加参数一起选择，然后用 output 命令导出。以下是有关此主题的一些实例。

选择一个时间范围

```
select * from PDE#HD#ProcessValueArchive#Analog where T between
        '1996-5-1 10:10:0.00' and '1996-6-1 10:10:0.00'
```

从时间标志开始选择

```
select * from PDE#HD#ProcessValueArchive#Analog where T >
        '1996-5-1 10:10:0.00'
```

按顺序和不按顺序选择过程值

```
select * from PDE#HD#ProcessValueArchive#Analog where V > 100
select * from PDE#HD#ProcessValueArchive#Analog where V > 100
        order by T
```

用列 T (时间)与 V (数值)来选择过程值

```
select T,V from PDE#HD#ProcessValueArchive#Analog where V > 100
        order by T
```

5.1.5 串行连接

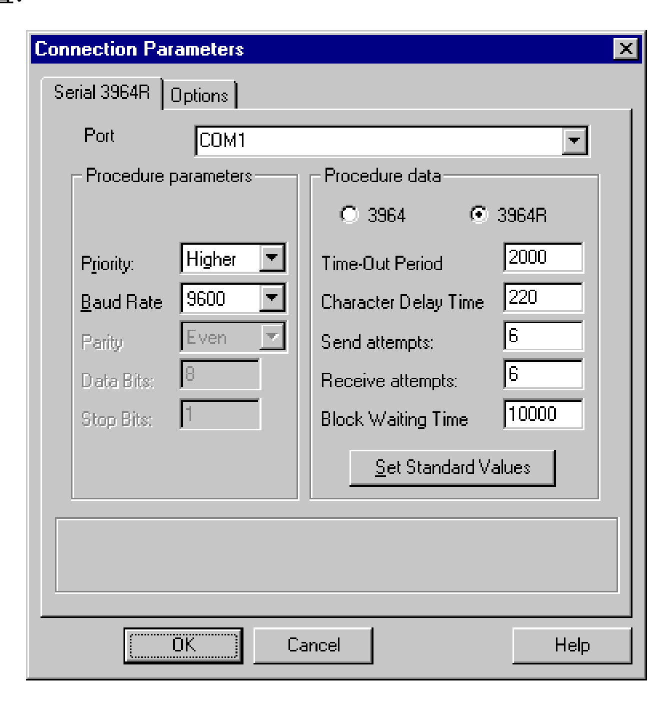
建立串行通讯连接，必须完成下列设置：

CP525 设置：

消息：	参数 CP525 名称：	P3964R
过程：	组件：RK	版本：01
波特率：	9600	字符长度：8
停止位数目：1	优先级：低	
奇偶校验：	偶数	

对于 PLC，需要在启动回路中为 CP525 设置 SYNCHRONOUS，在循环程序中设置 SEND/RECEIVE ALL。

WinCC 设置：



为了达到优化的目的，在两个伙伴中最好是 WinCC 具有*高优*先级。

5.1.6 颜色表

从一块大型调色板中组成颜色值。
十六种基本色是：

颜色	颜色值(十六进制)	符号常数
红色	0x000000FF	CO_RED
暗红色	0x00000080	CO_DKRED
绿色	0x0000FF00	CO_GREEN
暗绿色	0x00008000	CO_DKGREEN
蓝色	0x00FF0000	CO_BLUE
暗蓝色	0x00800000	CO_DKBLUE
青色	0x00FFFF00	CO_CYAN
暗青色	0x00808000	CO_DKCYAN
黄色	0x0000FFFF	CO_YELLOW
暗黄色	0x00008080	CO_DKYELLOW
绛红色	0x00FF00FF	CO_MAGENTA
暗绛红色	0x00800080	CO_DKMAGENTA
淡灰色	0x00C0C0C0	CO_LTGRAY
灰色	0x00808080	CO_DKGRAY
黑色	0x00000000	CO_BLACK
白色	0x00FFFFFF	CO_WHITE

符号常数用#define 进行外部预定义。
通过使用调色板颜色的中间值来产生混合颜色。
如果借助动态对话框创建了颜色改变，并且随后用 C 动作对所组态的数据进行处理，则即使颜色值是十进制格式同样能被读出。

5.2 S5 报警系统的文档

S5 报警系统的任务和功能

本文档描述了 SIMATIC S5 软件的功能和属性：
S5 报警系统

本软件用以确保顺序采集二进制消息并对其进行处理和缓冲。 程序包提供 SIMATIC S5 内所需的软件功能以实现 WinCC 系统的顺序采集消息的功能。

软件的主要运行可以说明如下：软件监控消息的二进制信号状态，用户可在消息界面中将其用于 S5 报警系统。如果信号条件改变，则可通过消息编号和日期/时间标志来识别消息。将一个 32 位过程变量和字母数字作业/批标识符添加到该数据中(如果用户对此进行了组态)。根据需要，将通过此方式组态的消息块缓冲到 FIFO 缓冲区中。如果每个时间单元所发出的消息要比使用现有的总线连接传送至 WinCC 系统的消息多，则需要对消息数据进行缓冲。此功能可使在 SIMATIC S5 中的顺序消息采集和高一级 WinCC 报警系统中的顺序消息采集之间，获得时间上的分离，并且使实时消息的处理成为可能。

由 S5 报警系统生成的消息块可在数据块界面中被 S5 应用程序所使用。通过使用 S5 通讯软件(必须由用户执行)，这些数据通过总线连接(例如，SINEC H1)传送到高一级的 WinCC 报警系统。在 WinCC 中，可以使用综合性的消息处理功能，如可视化、归档、报表等。
用户通过数据块接口(系统 DB 80)进行 S5 报警系统的组态。用户在此处确定运行报警系统的系统要求。在此处指定由 S5 报警系统所使用的存储区域、要处理的消息的类型、范围和地址区域的分配。

本节描述了 SIMATIC S5 环境下 S5 报警系统的应用和处理。提供了由软件所使用的功能块和数据块以及所需要的存储空间的概述。随后是对 S5 报警系统和 S5 应用程序间所有存在的数据接口的更深入的描述。还包括一个组态实例以方便开始了解 S5 报警系统。

5.2.1 列出软件块

SIMATIC S5 软件以及与该手册相关的实例一起位于光盘上，以 *WINCC1ST.S5D* 文件名存储。
该文件包含下列 S5 报警系统的功能和数据块：

功能块	名称	大小	功能
FB 80	SYSTEMFB	1114	序列报表
FB 81	STARTUPFB	135	序列报表的启动和初始化
FB82	PCHECK	574	由 FB 81 调用
FB 83	MBLOCK	699	由 FB 80 调用
FB 84	WRITE	94	由 FB 80 调用
FB 87	FULL	87	由 FB 80 调用
DB 80	System DB	512	将参数分配给报警记录
All		2703	

表 1

所需的最小存储空间取决于 S5 报警系统的组态。经常还需要下列数据块。

FIFO 缓冲器(最小)	*2 DB*	*= 1024 字节*
传送信箱至 WinCC	*1 DB*	*= 512 字节*

对每个偏移量数据块或参数数据块，另外还必须包括 512 个字节。
对偏移量数据块和参数数据块的大小的确切计算在偏移量数据块的结构和参数数据块的结构章节中进行了说明。

5.2.2 硬件要求

在表 1 中为 S5 报警系统指定的功能块需要下列硬件才能正确执行：

PLC	CPU
PLC 115U	CPU 944 *，CPU 945
PLC 135U	CPU 928B
PLC 155U	CPU 946/ 947，CPU 948

表 2

* 只有带两个 PG 接口的 CPU 944 才具有系统时钟。
这些 CPU 都具有一个内部时钟，可以允许它们提供生成消息块的当前日期/时间。
对于每个建立的 WinCC 通道，都周期性地将当前日期/时间消息写入 SIMATIC S5 CPU。SIMATIC S5 的内部时钟通过功能块 **FB 86：MESS:CLOCK** 与 WinCC 的系统时钟同步。

5.2.3 将 S5 报警系统集成到 SIMATIC S5 应用程序中

为了将用于报警系统的 SIMATIC S5 软件集成到 SIMATIC S5 应用程序中，必须执行下列步骤：
表 1 中指定的所有块必须从 WinCCST.S5D 文件传送至相应的 PLC。
如果在缺省情况下还未完成或者它们在 PLC 中不可用，则为相关的 PLC 传送数据处理块。

步骤	过程：集成报警系统
1	表 1 中指定的所有块必须从 WinCCST.S5D 文件传送至相应的 PLC。
2	如果在缺省情况下还未完成或者它们在 PLC 中不可用，则为相关的 PLC 传送数据处理块。
3	按照将参数分配给 DB80 一章，将参数分配给数据块 DB 80。
4	按照建立数据块一章，为发送信箱、FIFO 缓冲区、消息偏移量和消息参数(如果需要)建立数据块。
5	按照初始化偏移量数据块一章，初始化不同消息类别的偏移量数据块。
6	规定应用程序中各消息的过程变量、作业与批标识符。
7	在启动 OB (OB 20、OB 21、OB 22)中调用下列块： • SPA HTB ： SYNCHRON (相关 CPU 的数据处理块) • SPA FB 81 ： STARTUPFB
8	在 OB 1 中调用下列块： • 用于周期性处理消息 SPA FB 80 : SYSTEMFB • 由用户创建的功能块，用于将消息块传送至高一级的 WinCC 系统
9	按照后面的章节添加其它功能： • 通过 FB 86 : MESS:CLOCK 使日期和时间同步。

表 3

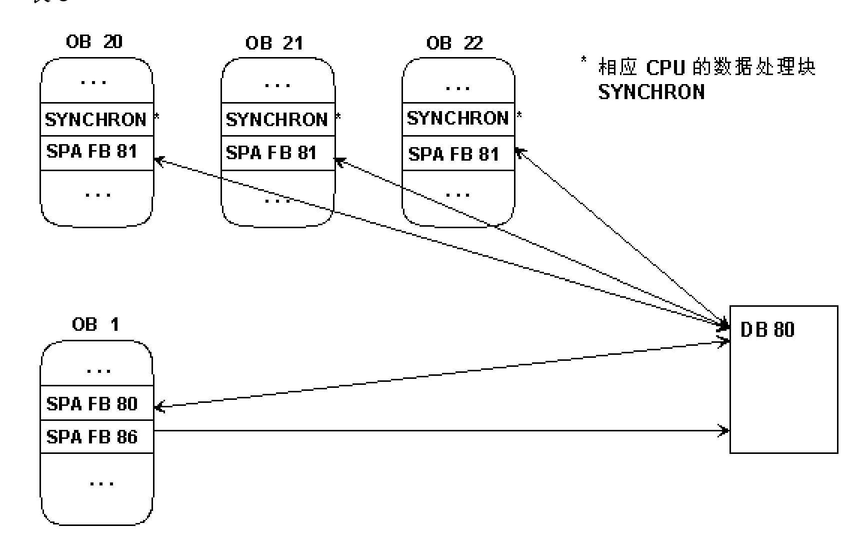


图 1

S5 报警系统的常规描述

将对 S5 报警系统的下列组件进行描述：

- 偏移量数据块
- 参数数据块
- 消息块
- FIFO 缓冲区
- 发送信箱
- 系统数据块

不同组件之间的关系：

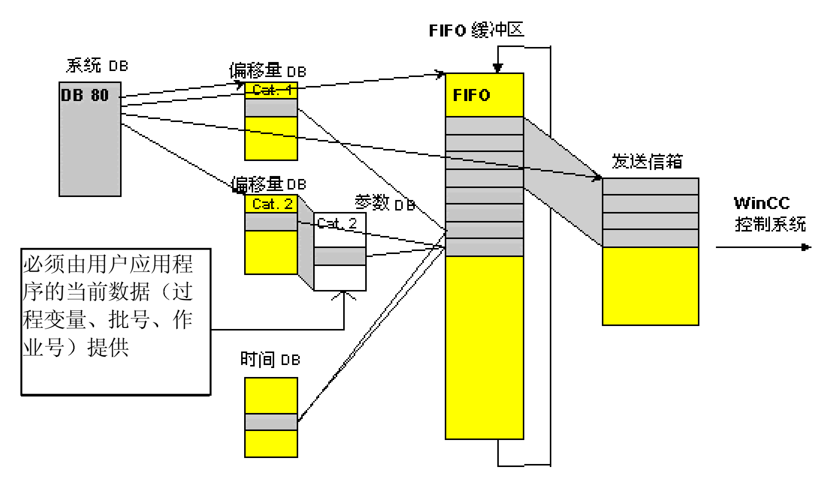


图 2

在消息采集系统能够监控和获取消息之前，消息必须在相应的数据块中进行组态。有四种不同的消息类别：

类别	定义：消息类别
1	不带参数的消息
2	带过程变量(2 个 DW)的消息
3	带过程变量(2 个 DW)和作业/批标识符(3 个 DW)的消息
4	带过程变量(2 个 DW)、作业/批标识符(3 个 DW)和保留(3 个 DW)的消息

表 4

对于报警系统，可以为消息块的生成全局指定日期/时间标志。如果日期/时间标志丢失，则由 WinCC 系统将相应的信息添加至消息块。

5.2.3.1 偏移量数据块的结构

偏移量数据块对全部四个消息类别具有相同的结构。在系统数据块 DB 80 中为每个所需的消息类别指定相应的数据块地址。
相应消息类别的偏移量数据块：

DW	内容	分配
DW 0	没有分配	标题
DW 1	基本消息编号	
DW 2	最后信号状态块的地址	
DW 3	没有分配	
DW 4	消息的信号状态 - 位编号： 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	信号状态块 1
DW 5	空闲状态位	
DW 6	确认位	
DW 7	边沿触发标记	
DW 8	消息的信号状态 - 位编号： 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	信号状态块 2
DW 9	空闲状态位	
DW 10	确认位	
DW 11	边沿触发标记	
DW 12	消息的信号状态 - 位编号： 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	信号状态块 3
DW 13	空闲状态位	

表 5

将描述下列元素：

- 基本消息编号
- 偏移量消息编号
- 信号状态块
- 最后信号状态块的地址
- 信号状态
- 空闲状态位
- 确认位
- 边沿触发标记

5.2.3.2 基本消息编号

每个消息被分配一个确定的消息编号，通过消息编号能够识别发出的消息。消息编号由基本消息编号和偏移量消息编号组成。
必须为使用的每种消息类别指定不同的基本消息编号。从该基本消息编号开始，这类消息通过偏移量消息编号来加以区分。
在相关的偏移量数据块 DW 1 中指定相应消息类别的基本消息编号。

特殊情况

如果使用消息类别 1，则可以使用两个偏移量数据块。为了使此消息类别的消息连续编号，必须将第一个偏移量数据块的基本消息编号加上本身的消息容量(1008 条消息)作为第二个偏移量数据块的基本消息编号输入。
计算消息编号：

消息编号 = 基本消息编号 + 偏移量消息编号

实例：

计算	描述：
给定：	消息类别 1，连续的消息编号始于：消息编号 10000
需要：	两个偏移量数据块的基本消息编号
10000	第一个偏移量数据块的基本消息编号
10000 + 1008 = 11008	第二个偏移量数据块的基本消息编号

5.2.3.3 偏移量消息编号/消息的信号状态

在相应消息类别的偏移量数据块中偏移量消息编号的各个位上包含了消息的信号状态。
相应消息的偏移量消息编号从 DW 4 的 16 位(位 0-15)开始。以 4 为增量进行连续编号(DW8、DW12 等)。

信号状态块	信号状态块的起始数据字	位编号 0 - 15 所对应的偏移量消息编号
1	4	0 - 15
2	8	16 - 31
3	12	32 - 47
4	16	48 - 63
...		...
62	248	976 - 991
63	252	992 - 1007

表 6

计算偏移量消息编号：

偏移量消息编号 = *消息编号 - 基本消息编号*
偏移量消息编号 = *(数据字 / 4 - 1) * 16 + 位编号(0-15)*
偏移量消息编号 = *(信号状态块 - 1) * 16 + 位编号(0-15)*

计算偏移量消息编号的 DB、DW 位编号：

数据块 = *偏移量数据块*
数据字 = *(偏移量消息编号 / 16 + 1) * 4*
位编号 = *偏移量消息编号 ? 16*

如果消息类别为 1，数据字的长度可能大于 252。则采用下列方式：

数据块 = *偏移量数据块 + 1*
数据字 = *数据字 - 252*
位编号 = *位编号*

实例 1：

假定：　　*DW 248，位 7，基本消息编号 = 10000*
需要：　　*消息编号*

信号状态块	=	*248 / 4*
	=	*62*
偏移量消息编号	=	*(信号状态块 - 1) * 16 + 位编号*
	=	*(62 - 1) * 16 + 7 = 983*
消息编号	=	*基本消息编号 + 偏移量消息编号*
	=	*10000 + 983 = 10983*

需要的消息编号是 10983。

实例 2：

假定：　　*具有两个偏移量数据块的消息类别 1，消息编号 = 12000，基本消息编号 = 10000*

需要：　　*DB、DW、位编号*

偏移量消息编号	=	*消息编号 - 基本消息编号*
	=	*12000 - 10000 = 2000*
位编号	=	*偏移量消息编号 ? 16 = 0*
数据字	=	*(偏移量消息编号 / 16 + 1) * 4*
	=	*(2000 / 16 + 1) * 4 = 504*

数据字大于 252。

数据块	=	*偏移量数据块 + 1*
数据字	=	*504 - 252 = 252*
位编号	=	*0*

可从消息类别 1、数据字 252、位编号 0 的第二个偏移量数据块中找到消息编号 12000。

5.2.3.4 信号状态块

第一个信号状态块从数据字地址 4 开始，随后的信号状态块以 4 个数据字为间隔(DW 8、DW 12 等)。
参见表 5 或表 6。
每个偏移量数据块允许 63 个信号状态块(信号状态块 1 至 63)。
一个信号状态块包含 16 个信号状态。因此在偏移量数据块中就可能有 63 * 16 = 1008 条消息。

信号状态块的结构：

DW	位编号	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	信号状态	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	空闲状态	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	确认位	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	边沿触发标志	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

表 7

在本章中提供了有关这 4 个位状态的附加信息。
计算相应的信号状态块：

信号状态块 = *(偏移量消息编号 / 16) + 1*

信号状态块 = *数据字 / 4*

计算相应信号状态块起始的数据字：
*信号状态块的第一个数据字 = 信号状态块 * 4*

5.2.3.5 最后一个信号状态块的地址

通过指定由消息占用的最后一个信号状态块的 DW 地址，指定相应消息类别的可能消息数。
计算最后一个信号状态块：
最后一个信号状态块 = 该消息类别所需的消息数 / 16

```
// Incompletely filled (16 messages) signal condition block
if ((required messages of this message category % 16) != 0)
{
        ++ last signal message block;
}
```

对于消息类别 1，产生的消息量可能超过 1008 条，以下是此情况的应用：

第一个偏移量 DB：

最后一个信号状态块 = 63

最后一个信号状态块的地址 = 63 * 4 = 252

第二个偏移量 DB：

最后一个信号状态块 = (该消息类别所需的消息数 - 1008) / 16

```
// Incompletely filled (16 messages) signal condition block
if (((required messages in this message category - 1008) % 16) != 0)
{
        ++ last signal message block;
}
```

计算最后一个信号状态块的 DW 地址：

*最后一个信号状态块的 DW 地址 = 最后一个信号状态块 * 4*

实例：

给定： 消息类别 1 的 1030 条消息

第一个偏移量 DB：

最后一个信号状态块的地址

第二个偏移量 DB：

所需的消息数 - 1008	=	*1030 - 1008 = 22*
(所需的消息数 - 1008) / 16	=	*22 / 16 = 1*
(所需的消息数 - 1008) ? 16	=	*22 ? 16 = 6*
最后一个信号状态块	=	*2*
最后一个信号状态块的地址		

5.2.3.6 信号状态

位置：信号状态块的第一个数据字(参见表 5)。
用户必须确保将相关消息的信号状态输入由相应消息类别的偏移量数据块所提供的数据字中。这可以由控制程序通过连续的伴随过程的信号更新来实现。

5.2.3.7 空闲状态

位置：信号状态块的第二个数据字(参见表 5)。
信号的空闲状态表示在被动操作状态时的信号级别。它定义了信号(消息)是被动还是主动。需要该信息以得知是到达一条消息还是发出一条消息。
如果事件的改变具有与空闲状态相反的状态，则是到达了一条消息。如果发出一条消息，则事件改变的状态与相关的空闲状态完全相同。
需要由用户在相应位置处指定消息的空闲状态。

5.2.3.8 确认位

位置：信号状态块的第三个数据字(参见表 5)。
在运行程序中确认位不进行组态而进行判断。此处，根据相应的确认原理由高一级的 PC 直接确认消息。PC 将这些与消息相关的确认连同集成的消息采集系统的组态消息发送到 PLC。
每个 PLC 周期由 S5 报警系统对相应的确认位进行设置。
应用程序必须相应地判断此信息。

5.2.3.9 边沿触发标记

位置：信号状态块的第四个数据字(参见表 5)。
边沿触发标记用于确定可能已经发生的事件的改变(消息的改变)。在 S5 报警系统中不能组态却能解释它们。

5.2.3.10 参数数据块的结构

对于消息类别 2 至 4，除了偏移量数据块以外，还要为相关消息的附加数据组态参数数据块。消息的信号状态存储在偏移量数据块中。参数数据块的地址存储在连续的数据块中，并且直接与相应的偏移量数据块相连。
偏移量数据块和参数数据块之间的关系：

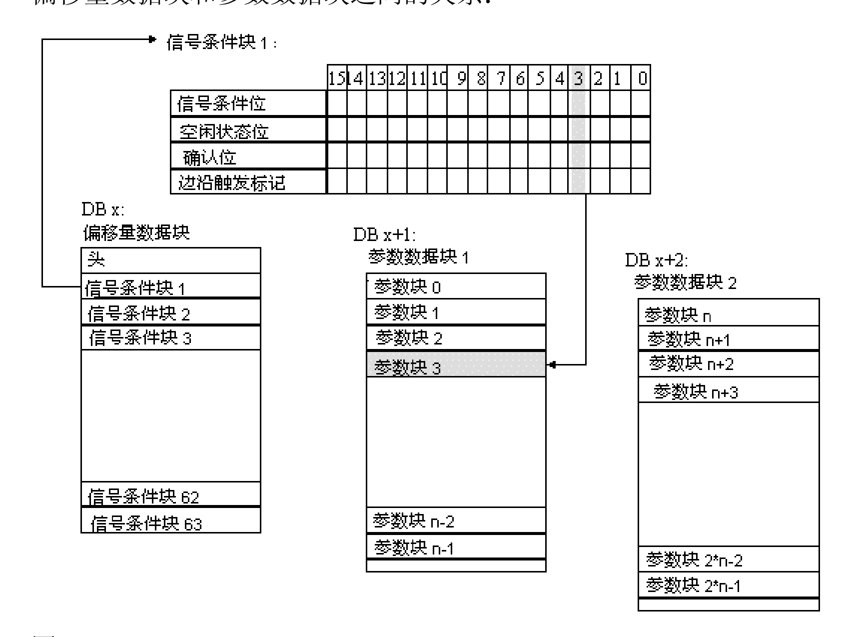


图 3

类别	最大数目	参数块的大小	每个参数 DB 的块数	参数 DB 的最大数目
1	1008 / 2016	-	-	-
2	1008	2 个 DW	128	8
3	1008	5 个 DW	51	20
4	1008	7 个 DW	36	28

表 8

计算参数数据块的数目：

$$\text{参数数据块编号} = \frac{\text{已使用的消息}}{\text{每个参数数据块中等参数块的数目}}$$

组态时要注意，确保地址不与另一个消息类别的数据块重叠，并且参数数据块的数目可以应付将来的升级。
参数数据块包含分配给不同消息的参数块。从该消息类别的第一条消息的参数块开始，参数块连续存储在参数数据块中。参数块持续增加超出参数 DB 的限制。一旦到达参数 DB 的末尾，就继续从下一个编号的参数 DB 的 DW 0 位置开始存储参数块。直到全部参数块存入参数数据块为止。
计算参数块的起始地址：

偏移量消息编号	=	消息编号 - 基本消息编号
参数 DB	=	偏移量 DB + 1 + (偏移量消息编号 / 每个参数 DB 的参数块数)
参数 DB 的起始地址	=	(偏移量消息编号 ? 每个参数 DB 的参数块数) * 参数块的大小

用户必须确保在相应地址处的相关数据(过程变量、作业号、批标识符)可用。

5.2.3.11 消息块的结构

发送到高一级 WinCC 系统的消息块由多个连续的数据字组成。数据字包含指定消息的信息。这些数据字合并为一个消息块。消息块的大小在各个消息类别中各不相同。

不论是哪个消息类别，消息块至少包含两个数据字。这就是消息编号和消息状态数据字。消息块最大可长达 12 个数据字，这主要取决于消息是否为日期/时间标志(三个数据字)以及是否具有合适的参数。

DW	描述：
第一个 DW	消息编号
第二个 DW	消息状态
第三个 DW	时间
第四个 DW	时间
第五个 DW	日期
第六个 DW	过程变量
第七个 DW	过程变量
第八个 DW	作业号
第九个 DW	作业号
第十个 DW	批标识符
第十一个 DW	保留
第十二个 DW	保留

表 9

如果消息不是日期/时间标志，则在第三到第五个位置的三个数据字可以不用。参数数据字可直接连到状态数据字。消息块的特定大小(DW 数)根据消息类别和所期望的日期/时间标志而不同，具体情况可以参考表 10。

根据消息类别来决定消息块的长度：

类别	DW 中没有日期和时间的消息块长度	DW 中具有日期和时间的消息块长度
1	2	5
2	4	7
3	7	10
4	9	12

表 10

5.2.3.12 消息编号

为每条消息分配一个确定的消息编号以便能明确地识别消息。

5.2.3.13 消息状态

消息状态的结构如下：

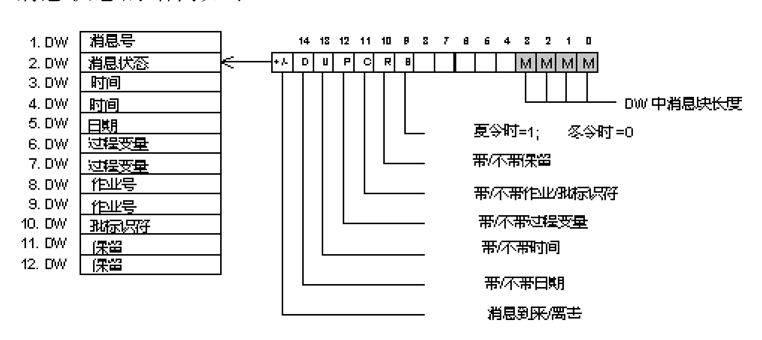


表 11

5.2.3.14 日期/时间标志

通过功能块 FB 86 : MESS:CLOCK 可使日期和时间以二进制代码形式使用。

5.2.3.15 过程变量

过程变量可用两个数据字来记录，如果到达一条消息，则将其转发给过程系统。

5.2.3.16 作业号/批标识符

根据组态，前两个数据字必须表示为有符号的 32 位二进制数或四个 ASCII 字符。第三个数据字必须为两个 ASCII 字符。
通过这三个数据字，在消息到达时可以将当前作业号或批标识符传送给 WinCC 系统。

5.2.3.17 保留

消息类别 4 的两个保留数据字用于将来的扩充，但是当前在 WinCC 系统中还未实现。

5.2.3.18 消息块的生成

在检测到消息之后，当前选中位的位置用于确定相应的消息编号，该编号作为消息块的第一个数据字存储在 FIFO 缓冲区中。根据消息的到达或消失、类别以及日期/时间标志的要求，选择相应的状态屏蔽并作为消息块的第二个数据字存储在 FIFO 缓冲区中。如果已经为日期/时间标志将参数分配给系统数据块中相应的位，则系统数据块 80 中可用的三个数据字(从地址 DW 190 开始)将采用所请求的 PC 格式。根据消息类别，如果必要则从相应的数据归档(参数数据块)中读取相关的参数块，并将其添加至 FIFO 缓冲区中的最后一个输入，以完成消息块。
然后，检查消息的下一个状态位。这一直继续进行，直到处理完所有已经分配了

参数的消息为止。

5.2.3.19 内部 FIFO 缓冲区(环形)

FIFO 缓冲区是一种呈环形的存储类型，也就是说存储环首尾相连。一方面，这将导致存储区大小受到限制；另一方面，由于一旦存储区填满它就重头开始存储，所以其大小也是无限的。
在消息采集系统中，这意味着一旦到达虚拟存储区的末尾(缓冲区满)，最旧的数据就会被最新的数据所覆盖。如果没有将先前的数据移走，就会因此而导致信息丢失。
顾名思义，RAM 中的 FIFO 缓冲区用作在采集消息转送至 PC 之前存放它们的缓冲区。在 RAM 中，FIFO 缓冲区由至少含有两个数据块的存储区组成，并且可根据参数分配组态为任意大小，只要它在 PLC 中所允许的最大数据块数目之内或在应用程序的剩余可用 DB 数目之内。用户把可由它用于归档的数据块数目传送给报警系统。
对于一个以上的数据块，必须使用具有连续 DB 编号的数据块。因此，用户应将缓冲区的起始 DB 编号和结束 DB 编号指定为系统 DB 中的参数。其值位于起始数据块和结束数据块之间的所有数据块(包括这两个数据块本身)均属于用作存储空间的缓冲区。

5.2.3.20 发送信箱 - 将数据传送给更高级的 WinCC 系统

通常，首先将当前周期的所有消息条目写入到 S5 报警系统的内部 FIFO 缓冲区中。
一旦采集完成，只要发送信箱已准备就绪，就将消息条目(最多为一个数据块内容的最大值)传送到消息接口(发送信箱)。数据块形式的消息界面作为传送功能块的数据源(STEP 5 - 数据处理块)。数据处理块为正被使用的过程总线(例如 SINEC-H1 总线)形成一个到相应通讯处理器的接口。

发送信箱的结构

DW	内容
DW 0	数据块的长度
DW 1	KY = [PLC 编号] , [CPU 编号]
DW 2	KY = [0] , [消息数目]
DW 3	用户数据(消息块)的启动

表 12

DW 0：

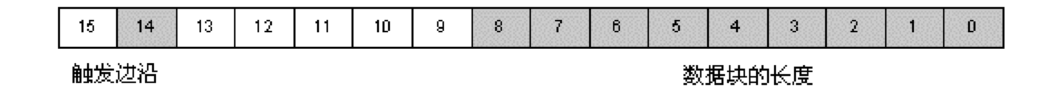


表 13

发送信箱的 DW 0 首先由位编号 14 确定(激活所请求作业的边沿触发)，然后由位编号 0 - 8 确定(源数据的长度)。
由于要传送的数据块最大可为 256 个数据字长，但是一个字节只能表示一个最大为 255 的数，因此使用 DL 或 DR 命令单独对字节进行查询是不可能的。因此，建议通过辅助标记来传送 DW 0。其优点是可以独立和直接地解释允许位。使用数据字时，不能使用该操作。
如果条件满足，应该对用作发送作业一次触发边缘的位进行重新设置。这样，其余的设置位就对应于所传送的源数据长度，且可将其作为 QLAE 写入间接参数分配的数据区中。
在成功完成对 WinCC 系统的写入请求(SINEC-H1)后(即无故障(FOF))，必须使用数值 0 对发送信箱的 DW 0 进行覆盖。这样可再次激活发送信箱和其它消息块，如果它们存在，可将其从内部 FIFO 缓冲区传送到发送信箱中。
必须使用 SEND 直接函数来实现写入请求(SINEC-H1)。与该函数有关的信息请参见相应的 PLC 手册。

5.2.4 接口说明

将对下列接口和块进行描述：

- 系统数据块 DB 80：用于将参数分配给 S5 报警系统。
- 用于相应消息类别的偏移量数据块：消息信号与具有消息属性规定的 S5 报警系统的二进制接口。
- 用于相应消息类别的参数数据块：用于指定类别 2 至类别 4 的附加消息数据。
- 发送信箱：与 WinCC 系统的传送接口。

5.2.4.1 系统数据块 80

通过使用系统数据块 DB 80，可以对用于四个消息类别的独立数据区、FIFO 存储器和发送信箱进行组态。为了进行组态，提供了 DB 80 中 0 到 20 的数据字。

5.2.4.2 偏移量数据块

如果需要 S5 报警系统就对相应消息的信号状态进行判断，并根据它们构成相应的消息块。
用户必须确保...

- 在组态空闲状态期间，指定各个消息。
- 在运行 S5 应用程序期间，将消息状态写入到相应的信号状态位中。
- 需要时读取并判断相应的确认位。

5.2.4.3 参数数据块

对于消息类别 2 至 4，可通过消息块传送与系统当前状态相关的附属消息。
用户必须确保...

- 一旦消息到达，相应的参数块中必须存在有效的过程变量(过程值、作业和批号)。

5.2.4.4 发送信箱/传送信箱

一旦发送信箱含有消息块，就通过写作业(SINEC-H1)直接将其传送给 WinCC 系统。
用户必须确保...

- 各 CPU 的相应数据处理块可用。
- 在组态 WinCC 系统期间，指定用于过程总线连接的合适的通讯通道。
- 触发一个写作业。

5.2.5 分配参数给 S5 报警系统/系统 DB 80

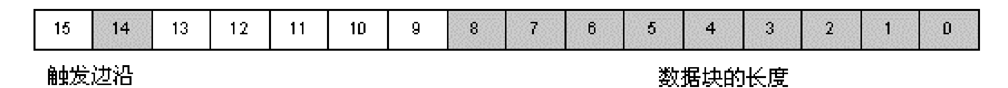
系统数据块 DB 80 的可组态数据字的描述：

DW	描述
0	DB 地址：内部 FIFO 起始
1	DB 地址：内部 FIFO 结束
2	0：无日期和时间，1：有日期和时间
3	用于类别 1 消息的 DB 偏移量
4	1：类别 1 的一个 DB 偏移量，2：类别 1 的两个 DB 偏移量
5	用于类别 2 消息的 DB 偏移量
6	用于类别 3 消息的 DB 偏移量
7	用于类别 4 消息的 DB 偏移量
8	保留
9	保留
10	DB 地址：发送信箱 CPU -> PC
11	1：优化采集(ACOP)
12	从 n 条消息启动的 ACOP
13	PLC 类型(115/135/155)
14	保留(必须是 1)
15	PLC 编号：1 至 255；CPU 编号：1 至 4
16	保留
17	保留
18	保留
19	保留
20	真实性检查的奇偶校验错误

表 14

DW 0、DW 1：内部 FIFO 缓冲区的 DB 存储区域
通过这两个数据字可对用于消息的内部 FIFO 缓冲区区域进行定义。
存储空间必须至少为两个数据块的大小，并且必须确保 FIFO 的结束参数大于 FIFO 的起始参数。
从数据块区域(包括两个指定数据块)生成的缓冲区的存储器区域受到 FIFO 起始地址和 FIFO 结束地址的限制。
FIFO 缓冲区大小的选择：
当达到 FIFO 缓冲区的存储容量时，最旧的消息将被覆盖。所选择的 DB 数必须足够大，以便在大量消息不断进入时，不会覆盖任何尚未导出的消息。为确保这点，应用下列从经验得出的规则：
确定 FIFO 缓冲区中的 DB 数。
每个 DB 的消息 = (255 DW / DB) / 消息块长度

参见表 10



对于消息采集系统的 *ACOP* 操作，建议添加一个或两个以上的数据块。

DW 2：日期和时间标识符

日期/时间标志的消息选项指的是所有已参数化的消息。获得的所有消息可以都有日期/时间标志(DW2 = 1)或一个也没有(DW2 = 0)。如果设置 DW2 = 0，则 WinCC 系统将添加一个日期/时间标志给到达的消息块。

DW 3、DW 4：消息类别 1 的偏移量 DB

如果要对类别 1 消息(不带参数和批标识符的消息)进行组态，则必须在数据字 3 中指定偏移量数据块的地址。必须由控制程序将这些消息的信号状态连续地写入到所指定的数据块中。

如果规划了类别 1 的 1008 条以上的消息(不超过 2016 条消息)，则通过在数据字 4 中输入 2 可激活用于类别 1 消息的附加数据块。第二个数据块将自动接收下一个比 DW 3 中的地址更高的地址。如果达到了类别 1 消息的最大值 1008，则在 DW 4 中输入 1。

DW 5、DW 6、DW 7：消息类别 2、3、4 的偏移量 DB

与 DW 3 类似，数据字 5 至 7 包含了存储消息信号的数据块地址。

DW 5 包含了用于消息类别 2 的数据块地址，而 DW 6 和 DW 7 则分别包含了消息类别 3 和 4 的地址。

如果消息类别没有使用，则必须在对应的 DW 中输入 0。

可将在 DW 5 至 DW 7 中所指定的地址称为偏移量 DB。根据消息类别以及每个类别的消息数，可为其分配一个相应的辅助 DB 数。辅助 DB 包含了消息的参数。因此，在分配偏移量 DB 地址时，必须确保在前一个偏移量 DB 与所指定的偏移量 DB 之间为参数 DB 分配了足够的空间(数据块)。

最多可为消息类别 2 至 4 组态 1008 条消息。当这些消息全部使用时，将根据不同的消息类别产生对应于偏移量 DB 的辅助 DB (参数 DB)的不同数目(参见表 8)。

DW 10：与更高一级 WinCC 系统的消息接口

必须始终为数据字 10 分配参数(由消息采集系统所使用的操作模式与此没有关系)。在 DW 10 中对传送信箱的 DB 地址进行了分配。传送信箱将被用作 SIMATIC S5 与更高一级 WinCC 系统之间的一个接口。

DW 11、DW 12：用于优化采集及相应消息数的模式选择

可使用两种操作模式：

- DW 11 中为 0 -> 消息采集系统的普通模式
- DW 11 中为 1 -> 消息采集系统的优化采集模式

普通模式：

在一个周期内采集并存储在内部缓冲区中的所有消息均通过消息接口(具有容量限制)发送给 WinCC 工作站，只要消息接口已准备接收数据。
如果在一个周期内或在几个连续的周期内到达了大量的消息，则该过程将花费相对较长的周期时间。周期时间还将随有关消息类别的消息块大小的增加而增加。在这种情况下，消息块的采集将更费力，所用的时间也更长。

优化采集：

对出现的消息按时间顺序进行采集要比发送至 WinCC 工作站优先。值得注意的是系统消息出现的相对时间间隔。消息可能要在几毫秒之后才在 WinCC 工作站上显示，这不是特别重要。人眼的较缓慢反应和在控制室中观察者的易于接受性才是决定因素。
为了减少消息采集系统在这种临界时间情况下的周期时间，引入了优化采集模式下的系统运行选项。在 DW 12 中指定 OB1 周期内所出现消息的最小数目。如果在当前的 OB1 周期期间，消息数超出了该最小数，则仅对消息进行采集和缓冲。在该 OB1 周期内，不会将消息送出或随后发送给通讯伙伴。
DW 15：PLC/CPU 编号
生成消息头时需要该数据字，并需要指定与项目相关的 PLC 编号和其 CPU 编号。如果在单台 PLC 中运行多个 CPU，则 CPU 编号就特别重要。只有与包含消息 ID 的数据字一起，更高一级的 WinCC 系统才能够把所发送的数据翻译为消息，以及分配指定消息的消息文本并对其进行相应解释。
DW 15 是在组态期间接收 S5 数据格式 KY 的唯一数据字，也就是说，可分别表示两个字节(使用逗号隔开)。左字节包含了 PLC 编号，它可在 1 至 255 之间。而在右字节中指定了 CPU 编号，它可在 1 至 4 之间。

实例：

KY	=	*10,2*
PLC 编号	=	*10*
CPU 编号	=	*2*

DW 20：参数分配错误

在 S5 报警系统启动时，对系统 DB 中已分配参数的所有数据字进行真实性检查。在这种情况下，对超出可能的数值范围、对已分配参数的数据块进行重叠或多重分配以及不符合规定等情况进行区分。
至于数据字格式的输出参数，该功能块使用了一个所谓的 PAFE 字(参数错误字)，它类似于系统指定的数据处理块。PAFE 字的状态可从系统 DB 80 的 DW 20 中取出。可对该 PAFE 字进行检查，在程序从 FB 81 返回后是否产生了错误。随后，即可采取相应的操作。
建议在 PAFE 字不为 0 时，让 PLC 跳转到停止状态。如果忽略 PAFE 字，则在程序执行时不能保证错误不会出现。

对 PAFE 字的解释

如果在出现错误后(PAFE 字不等于 0)，程序或 PLC 如所建议的那样处于其停止状态，则可根据出错代码对错误进行分析和专门纠正。下表提供了参数分配期间可能引起的有关错误类型的信息。

PAFE 字的格式：

KY = *出错代码，组出错 ID*

实例：

KY = 9,1

编号为 9 的参数分配错误对应于：
类别 1 的偏移量 DB 地址大于最大允许的 DB 地址。

出错代码	描述
1	内部缓冲区的起始 DB 没有定义
2	内部缓冲区的起始 DB 具有和系统 DB (80)相同的地址
3	内部缓冲区的起始 DB 地址大于最大允许的 DB 地址
4	内部缓冲区的结束 DB 具有和系统 DB (80)相同的地址
5	结束 DB 地址小于内部缓冲区的起始 DB 地址
6	内部缓冲区的结束 DB 地址大于最大允许的 DB 地址
7	类别 1 的偏移量 DB 具有和系统 DB (80)相同的地址
8	类别 1 的偏移量 DB 地址在内部缓冲区中
9	类别 1 的偏移量 DB 大于最大允许的 DB 地址
10	类别 2 的偏移量 DB 具有和系统 DB (80)相同的地址
11	类别 2 的偏移量 DB 和类别 1 的偏移量 DB 具有相同的地址
12	类别 2 的偏移量 DB 和类别 1 的第二个偏移量 DB 具有相同的地址
13	类别 2 的偏移量 DB 地址在内部缓冲区中
14	类别 2 的偏移量 DB 大于最大允许的 DB 地址
15	类别 3 的偏移量 DB 具有和系统 DB (80)相同的地址
16	类别 3 的偏移量 DB 具有和类别 1 的偏移量 DB 相同的地址
17	类别 3 的偏移量 DB 和类别 1 的第二个偏移量 DB 具有相同的地址
18	类别 3 的偏移量 DB 和类别 2 的偏移量 DB 具有相同的地址
19	类别 3 的偏移量 DB 地址在内部缓冲区中
20	类别 3 的偏移量 DB 大于最大允许的 DB 地址
21	类别 4 的偏移量 DB 具有和系统 DB (80)相同的地址
22	类别 4 的偏移量 DB 和类别 1 的偏移量 DB 具有相同的地址
23	类别 4 的偏移量 DB 和类别 1 的第二个偏移量 DB 具有相同的地址
24	类别 4 的偏移量 DB 地址在内部缓冲区中
25	类别 4 的偏移量 DB 大于最大允许的 DB 地址
26	类别 4 的偏移量 DB 和类别 2 的偏移量 DB 具有相同的地址

出错代码	描述
27	类别 4 的偏移量 DB 和类别 3 的偏移量 DB 具有相同的地址
28	PC 发送信箱具有和系统 DB (80)相同的地址
29	没有定义 PC 发送信箱(0)
30	PC 发送信箱地址在内部缓冲区中
31	PC 发送信箱的地址大于最大允许的 DB 地址
32	PC 发送信箱和类别 1 的偏移量 DB 具有相同的地址
33	PC 发送信箱和类别 2 的偏移量 DB 具有相同的地址
34	PC 发送信箱和类别 3 的偏移量 DB 具有相同的地址
35	PC 发送信箱和类别 4 的偏移量 DB 具有相同的地址
36	PC 发送信箱和类别 1 的第二个偏移量 DB 具有相同的地址
37	保留 DW 9 或保留 DW 10 不等于 0
38	保留 DW 9 或保留 DW 10 不等于 0
39	保留 DW 9 或保留 DW 10 不等于 0
40	保留 DW 9 或保留 DW 10 不等于 0
41	保留 DW 9 或保留 DW 10 不等于 0
42	保留 DW 9 或保留 DW 10 不等于 0
43	保留 DW 9 或保留 DW 10 不等于 0
44	保留 DW 9 或保留 DW 10 不等于 0
45	保留 DW 9 或保留 DW 10 不等于 0
46	保留 DW 9 或保留 DW 10 不等于 0
47	用于对具有优化采集功能的所选操作模式进行最小限制的消息数丢失
48	没有定义 PLC 类型
49	保留 DW 14 不等于 1
50	没有定义用于消息头的 PLC 编号
51	没有定义用于消息头的 CPU 编号
52	CPU 编号大于允许值(1 到 4)

表 15

5.2.6 S5 报警系统的组态实例

说明

为下列消息类别组态 S5 报警系统：

类别	定义：消息类别
1	1200 条消息(从消息编号 10000 到 11199)消息 11000 至 11199 是低位激活。
2	没有计划的消息
3	11 条消息(从消息编号 30000 至 30010)
4	没有计划的消息

消息全部具有日期/时间标志。
将使用一个具有 PLC 编号 1 和 CPU 编号 1 的 135U。

5.2.6.1 DB 80 参数化

类别	最大数目	参数块的大小	每个参数 DB 的块数	参数 DB 的最大数目
1	1008 / 2016	-	-	-
2	1008	2 个 DW	128	8
3	1008	5 个 DW	51	20
4	1008	7 个 DW	36	28

将 DB 81 用作 PC 发送信箱。
由于现有消息的均匀出现，平均消息块长度(带有日期和时间)为：*(1200 * 5 + 11 * 10) / (1200 + 11)=5.05*
假定：
S5 报警系统将在一个 PLC 周期内提供具有 100 条消息的消息浪，并从 30 条消息开始在优化采集模式下运行。

*5 DW/消息*100 条消息 = 500 条 DW*

(500 DW)/(256 DW/DB) = 1.95 DB

这将产生用于 FIFO 缓冲区的四个数据块，因为必须添加一至二个以上的数据块用于优化采集模式。
FIFO 缓冲区起始地址为 DB 82，这样，FIFO 缓冲区的结束地址就为 DB 85。
为了给 FIFO 缓冲器的进一步扩展留有余地，把 DB 88 和 DB 89(用于 1008 条以上的类别 1 消息)作为类别 1 的偏移量数据块。
DB 90 作为消息类别 3 的偏移量数据块。消息类别 3 的参数 DB 容纳的参数块可多达 51 个；如果减去所使用的 11 个消息块，结果将导致一种对仅带有一个参数数据块(DB 91)的 40 条类别 3 消息的可扩展性。

DW	描述	数值
0	DB 地址：内部 FIFO 起始地址	82
1	DB 地址：内部 FIFO 结束地址	85
2	0：没有日期和时间 1：具有日期和时间	1
3	用于类别 1 消息的 DB 偏移量	88
4	1：类别 1 的一个 DB 偏移量 2：类别 1 的两个 DB 偏移量	2
5	用于类别 2 消息的 DB 偏移量	0
6	用于类别 3 消息的 DB 偏移量	90
7	用于类别 4 消息的 DB 偏移量	0
8	保留	0
9	保留	0
10	DB 地址：发送信箱 CPU -> PC	81
11	1：优化采集(ACOP)	1
12	从 n 消息开始的 ACOP	30
13	PLC 类型(115/135/155)	135
14	保留	1
15	PLC 编号：1 至 255；CPU 编号：1 至 4	1，1
16	保留	0
17	保留	0
18	保留	0
19	保留	0
20	真实性检查的奇偶校验出错	0

通过 DW 10 至 DW 20 使用数据块 100 来同步时间。
通过 DW 0 至 DW 255 使用数据块 101 来接受命令。

5.2.6.2 数据块的建立

创建数据块 DB 81 - DB 85、DB 88 - DB 91 和 DB101 的 DW 0 - DW 255。
创建数据块 DB 100 的 DW 0 - DW 20。

5.2.6.3 偏移量数据块的初始化

消息类别 1
为消息类别 1 提供 DB 88 和 DB 89。DB 88 包含编号从 10000 至 11007 的消息，DB 89 包含编号从 11008 至 11199 的消息。
总共要组态 1200 条类别 1 的消息。
参见最后一个信号状态块的地址一章
偏移量消息编号 = 消息编号 - 基本消息编号 = 0 至 1199

第 1 个偏移量 DB：

最后一个信号状态块的地址：DW 252

第 2 个偏移量 DB：

最后一个信号状态块的地址：DW 252

1200 - 1008	=	*192*
192 / 16	=	*12*
192 ? 16	=	*0*
偏移量数据块 2 中的最后一个信号状态块的地址	=	*12 * 4 = 48*

DB 88：

DW	描述	数值
DW 0	未分配	
DW 1	基本消息编号	10000
DW 2	最后一个 DW 的地址	252
DW 3	未分配	

DB 89：

DW	描述	数值
DW 0	未分配	
DW 1	基本消息编号	11018
DW 2	最后一个 DW 的地址	48
DW 3	未分配	

参见偏移量消息编号/消息的信号状态一章
消息 11000 至 11199 是低位激活的。
消息编号 11000 的空闲状态位的位置：

偏移量消息编号：	*11000 - 10000 = 1000*
信号状态块的起始地址	*(偏移量消息编号 / 16 + 1) * 4 = (62 + 1) * 4 * DW252*
空闲状态位的数据字：	*DW 253*
数据位：	*偏移量消息编号 ? 16 = 8*
数据块：	*偏移量数据块 = DB 88*

消息编号 11000 的空闲状态位的位置：

偏移量消息编号： *11199 - 10000 = 1199*

信号状态块的起始地址： *(偏移量消息编号 / 16+ 1) * 4 = (74 +1) * 4 = 300*

300 - 252 =*48*

空闲状态位的数据字： *DW 49*

数据位： *偏移量消息编号 ? 16 = 15*

数据块： *偏移量数据块 + 1 = DB 89*

下列空闲状态位必须进行修改：

DB 88：

DW 253：将数据位 8 至 15 设置为 1

DB 89：

DW 5、DW 9、DW 13 至 DW 49：将数据位 0 至 15 设置为 1
消息类别 3
对于消息类别 3，提供 DB 90 作为消息 30000 至 30010 的偏移量数据块，而 DB 91 则作为其参数数据块。
总共要组态 11 条类别 3 的消息。
参见参数数据块的结构一章
偏移量消息编号 = 消息编号 - 基本消息编号 = 0 至 10

偏移量 DB：

最后一个信号状态块的地址 *11 ／ 16 ＝ 0*

11 ? 16 ＝ 11

最后一个信号状态块的地址 = *(0+1) ＊ 4 ＝ 4*

DB 89：

DW	描述	数值
DW 0	未分配	
DW 1	基本消息编号	30000
DW 2	最后 DW 的地址	4
DW 3	未分配	

所有空闲状态位均为 0。
参见参数数据块的结构一章

参数 DB

消息编号 30000：

参数 DB = 90 + 1 + 0 / 51 = 91

消息编号 30010：

参数 DB = 90 + 1 + 10 / 51 = 91

相应参数块的起始地址	=	(偏移量消息编号 ? 每个参数 DB 的参数块数目) * 参数块的大小

消息编号 30000：

相应参数块的起始地址	=	*(0 ? 51) * 5 = DW*

消息编号 30010：

相应参数块的起始地址	=	*(10 ? 51) * 5 = DW 50*

DB 91：对应偏移量数据块 90 的参数数据块 91

消息编号	过程值	作业号	批标识符
30000	DW 0、1	DW 2、3	DW 4
30001	DW 5、6	DW 7、8	DW 9
30002	DW 10、11	DW 12、13	DW 14
30003	DW 15、16	DW 17、18	DW 19
30004	DW 20、21	DW 22、23	DW 24
30005	DW 25、26	DW 27、28	DW 29
30006	DW 30、31	DW 32、33	DW 34
30007	DW 35、36	DW 37、38	DW 39
30008	DW 40、41	DW 42、43	DW 44
30009	DW 45、46	DW 47、48	DW 49
30010	DW 50、51	DW 52、53	DW 54

5.2.7 SIMATIC S5 命令块的文档

S5 命令块的目的和功能

软件用于通过过程总线(即工业以太网)对 SIMATIC S5 中的位、字节、字以及双字进行处理。通过过程总线，只可能对 SIMATIC S5 中的字节或字的值进行寻址。

缺省状态下，可以执行下列操作：

- 必须仅按字对数据块(DB 和 DX)、定时器和计数器进行修改。
- 必须仅按字节对标志、输入、输出、外围设备(P 和 Q)进行修改。

程序包提供了 SIMATIC S5 内所需的软件功能度，以便通过给定的过程总线完成 WinCC 系统的下列操作：

- 为一个 OB1 周期设置起始时钟脉冲
- 对 DB/DX 中的位进行置位、复位、取反
- 对标志中的位进行置位、复位、取反
- 将左/右字节写入 DB/DX
- 将字/双字写入 DB/DX
- 将字节/字写入标志
- 将字节/字写入外围设备
- 将字节/字写入扩展外围设备

WinCC 控制中心可通过数据接口，以原始数据变量的形式在 SIMATIC S5 中进行期望的修改。命令必须通过该原始数据变量发送给 S5。通过指令解释器 FB 87: EXECUTE 可在 S5 中直接判断和执行这些命令。

本手册描述了 SIMATIC S5 环境中 S5 命令块的使用和处理。用户将对软件所使用的功能和数据块以及所需存储空间有一个较全面的了解。以下部分将对现有数据接口进行详细的描述。给出的组态实例能提供其它的帮助。

5.2.7.1 软件块的列表

SIMATIC S5 软件的 *S5 命令块*位于 WinCC 光盘中的名称为 **WINCC1ST.S5D** 的文件中。
该文件包含有用于 S5 命令块的下列功能块：

功能块	名称	字节大小	功能
FB 87	执行	152	允许通过过程总线激活位、字节、字和双字操作
FB 88	OPCODE	399	由 FB 87 调用
总计		551	

表 16

此外，还需要一个 512 字节的命令数据块。

5.2.7.2 硬件要求

为了能正确执行，表 16 中所指定的功能块需要下列硬件：

PLC	CPU
PLC 115U	CPU 943、CPU 944、CPU 945
PLC 135U	CPU 928A、CPU 928B
PLC 155U	CPU 946/947，CPU 948

5.2.7.3 FB 87: EXECUTE 的调用参数

下面对功能块 FB 87: EXECUTE 的调用参数进行了描述。

名称	执行	参数
ID：	DBNR	E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KF
ID：	DBDX	E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KF
ID：	RIMP	E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KY

DBNR：命令传送接口的数据块编号
DBDX：用于命令传送接口的数据源类型。
DB：数据源是一个数据块(DB)
DX：数据源是一个扩充数据块(DX)
RIMP：用于初始化脉冲的位的位置
RIMP：标志号、位号

5.2.8 接口说明

将对下列接口和数据块进行描述：

- 命令功能块 FB 87
- 命令数据块：与 SIMATIC S5 的命令传送接口

在 SIMATIC S5 中，命令解释器(FB 87: EXECUTE)将在 OB 1 内周期性调用。命令 DB 的类型和地址作为参数进行传送。当命令排队时，Op 代码和四个参数将被传送给 FB 88: OPCODE 并直接执行。在执行一条命令后，命令计数器(DW 1)减 1。重复进行命令的传送过程和命令计数器的减 1 过程，直到将排队命令全部处理完毕。

具体的数据块类型和地址必须与 WinCC 控制中心和 S5 程序二者保持一致，且 S5 中必须存在数据块。可供选择的是 DB 或 DX 数据块及其地址(即 DX 234)。用户需要打开数据块，由于数据块至多包含 255 个数据字，因此在指定的数据块中可以寻址数据字 0-255。

已为存储在命令数据块中的命令定义了下列语法：

DW	描述
0	没有使用
1	将执行的命令数
2	第一个命令的 Op 代码
3	参数 1 (Op 代码 1)
4	参数 2 (Op 代码 1)
5	参数 3 (Op 代码 1)
6	参数 4 (Op 代码 1)
7	第二个命令的 Op 代码
8	参数 1 (Op 代码 2)
9	参数 1 (Op 代码 2)
10	参数 2 (Op 代码 2)
11	参数 3 (Op 代码 2)
12	参数 4 (Op 代码 2)
13	第三个 Op 代码
14	参数 1 (Op 代码 3)
……	

执行命令的语法描述如下：

将 Op 代码和参数传送给命令 DB

命令	Op 代码	参数 1	参数 2	参数 3	参数 4
DB 中的置位	10	DB	DW	位	-
DB 中的复位	11	DB	DW	位	-
在 DB 中对位取反	12	DB	DW	位	-
在 DB 中对右字节进行设置	15	DB	DW	数值	-
在 DB 中对左字节进行设置	16	DB	DW	数值	-
写数据字到 DB	17	DB	DW	数值	-
写双字到 DB	18	DB	DW	数值	数值
DX 中的置位	20	DX	DW	位	-
DX 中的复位	21	DX	DW	位	-
DX 中取反	22	DX	DW	位	-
在 DX 中对右字节进行设置	25	DX	DW	数值	-
在 DX 中对左字节进行设置	26	DX	DW	数值	-
写数据字到 DX	27	DX	DW	数值	-
写双字到 DX	28	DX	DW	数值	数值
对标志位进行设置	30	MB	位	-	-
对标志位重新进行设置	31	MB	位	-	-
对标志位取反	32	MB	位	-	-
写标志字节	35	MB	数值	-	-
写存储器字	36	MW	数值		-
写外围设备字节	45	PB	数值	-	-
写外围设备字	46	PW	数值	-	-
写扩展外围设备字节	55	QB	数值	-	-
写扩展外围设备字	56	QW	数值	-	-
设置起始时钟脉冲	60	-	-	-	-

5.2.8.1 S5 命令块的组态实例

将建立 S5 命令块。
在标志字 56 的第 4 位中提供了初始化的脉冲。DX 237 则用作命令数据块。请确保在 PLC 中数据块 DX 237 打开 DW 0 至 DW 255。
在 WinCC 控制中心中，当指定通道参数(例如工业以太网)时输入所期望的数据块。

摘自 OB 1：

```
: SPA   FB 87
NAME  : EXECUTE
DBNR  : KF +237
DBDX  : KC DX
RIMP  : KY 56, 4
```

5.2.9 S5 时间同步的任务和功能

本文档描述了 SIMATIC S5 软件的功能和属性：

S5 时间同步

此软件用于使 SIMATIC S5 系统时钟同步。此外，还对 S5 报警系统的时间顺序消息采集提供了合适的日期/时间格式，以用于消息块的生成。
功能块 FB 86: MESS:CLOCK 也按照时间顺序消息采集所要求的格式提供了当前的 S5 时间。数据位于系统数据块 80 中从 DW 190 开始的部分。
如果消息信号状态发生改变，则功能块 FB 80: SYSTEMFB 将通过消息编号来识别消息，并用系统数据块 80 中的当前的日期/时间为其进行标记。
本手册详细描述了 SIMATIC S5 环境下 S5 时间同步的应用和处理。用户将对软件所使用的功能和数据块以及所需存储空间有一个较全面的了解。给出了一个组态实例以提供额外的帮助。

5.2.9.1 软件块的列表

SIMATIC S5 软件(S5 时间同步)位于 WinCC 光盘中名为 WINCC1ST.S5D 的文件中。

该文件包含下列功能和数据块：

FB 功能块	名称	大小(按字节计)	功能
FB 86	MESS:CLOCK	1135	时间的同步
总计		1135	

表 17

时钟数据区 115U:	*27 个 DW*	=	*54 个字节*
时钟数据区 135U/155U:	*12 个 DW*	=	*24 个字节*
用于 S5 报警系统的数据区:	*3 个 DW*	=	*6 个字节*

5.2.9.2 硬件要求

为了正确执行 S5 报警系统所指定的功能块，需要下列硬件：

PLC	CPU
PLC 115U	CPU 944 *，CPU 945
PLC 135U	CPU 928B
PLC 155U	CPU 946/947，CPU 948

* 只有带两个 PG 接口的 CPU 944 才具有系统时钟。

5.2.10 FB 86: MESS:CLOCK 的参数调用

下面对功能块 FB 86: MESS:CLOCK 的参数调用进行了描述。

名称	MESS:CLOCK	参数
ID：	CPUT	E/A/D/B/T/Z:D KM/KH/KY/KC/KF/KT/KZ/KG:KF
ID：	DCF7	E/A/D/B/T/Z:D KM/KH/KY/KC/KF/KT/KZ/KG:KF
ID：	QTYPE	E/A/D/B/T/Z:D KM/KH/KY/KC/KF/KT/KZ/KG:KF
ID：	QSYN	E/A/D/B/T/Z:D KM/KH/KY/KC/KF/KT/KZ/KG:KY
ID：	UDAT	E/A/D/B/T/Z:D KM/KH/KY/KC/KF/KT/KZ/KG:KY
ID：	ZINT	E/A/D/B/T/Z:D KM/KH/KY/KC/KF/KT/KZ/KG:KF
ID：	ZCLOCK	E/A/D/B/T/Z:D KM/KH/KY/KC/KF/KT/KZ/KG:KY
ID：	ZSYN	E/A/D/B/T/Z:D KM/KH/KY/KC/KF/KT/KZ/KG:KF

CPUT：

CPU 的编号	类型
1	CPU 943/CPU 944
2	CPU 945
3	CPU 928B
4	CPU 946/947
5	CPU 948

DCF7：

运行模式
0=以 S5 系统时钟运行
1=以 DCF77 无线电时钟运行

QTYPE：

用于时间同步消息的数据源类型
0=数据源为数据块(DB)
1=数据源为扩充的数据块(DX)

QSYN：

时间数据的数据源

DCF7 = 0：	QSYN =	时间同步消息的 DB 编号、DW 编号已收到
DCF7 = 1：	QSYN =	DCF77 时间的 DB 编号、DW 编号

UDAT：

时钟数据区地址
UDAT = DB 编号、DW 编号

ZINT：

按分钟发送同步消息的时间间隔(DCF7 = 1)

ZCLOCK：

报警系统格式下的时间数据的目标数据区
ZCLOCK = DB 编号、DW 编号

ZSYN：

时间同步消息的目标数据区(DCF7 = 1)
如果时间顺序消息采集功能度与 S5 报警记录一起使用，则需要一个从 DB 80 的 DW 190 字节开始的特殊时间数据格式。
从 S5 系统时间获取该时间数据格式，并将其写入到相应的 ZCLOCK 数据区(DB 80、DW 190 - 192)中。
时间顺序报表与 FB 86: MESS:CLOCK 之间的关系：

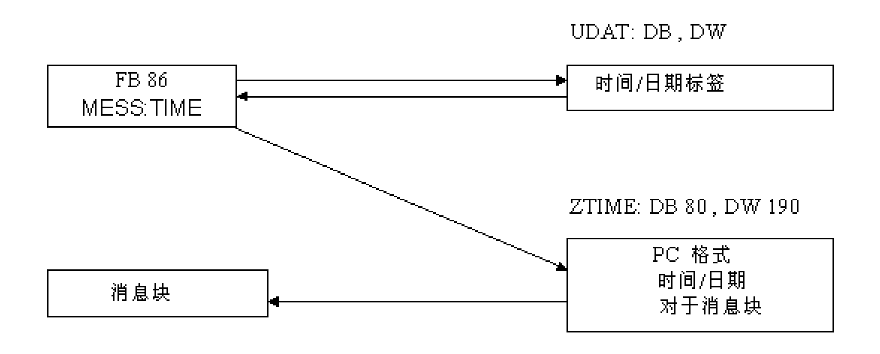


图 4

5.2.11 用于日期和时间的数据格式

系统的时间同步消息(WinCC 目前不支持时间消息)
时间同步消息帧的第一个数据字包含一个源 ID，它与日期和时间数据一起由系统进行发送。
仅当源 ID FFFF 定位于该位置之后，功能块 FB 86: MESS:CLOCK 才能获取未决的消息。通过该数据字中的 0 可对消息的收到进行确认。只有在新的消息到达(DW 1 = FFFF)后，才能再次对其进行读取和判断。

描述	数据字	内容	有效范围	注释
源 ID/时间消息	1	FFFF		
消息 ID	2	FFFF		没有使用
秒	3	00xx	xx：0 至 59	
分	4	00xx	xx：0 至 59	
小时	5	00xx	xx：0 至 23	
日	6	00xx	xx：1 至 31	
月	7	00xx	xx：1 至 12	
年	8	00xx	xx：0 至 127 (1990-2117)	年 + 1990
星期几	9	00xx	xx：0 至 6	星期日 = 0
一年中的第几天	10	00xx	xx：1 至 365	
夏令制，冬令制闰年	11	yyxx	xx：冬令制 = 00 夏令制 = 01 yy：闰年 本年度 = 00 上一年 = 01 两年前 = 02 三年前 = 03	

表 18

5.2.11.1 时钟数据区 CPU 944、CPU 945

对数据字编号的规定是相对的。区域的实际位置由调用参数：UDAT = FB 86: MESS:CLOCK 的 DB 编号、DW 编号来确定。

<table>
<tr><th>DW</th><th>位置</th></tr>
<tr><td>0</td><td rowspan="4">内部变量</td></tr>
<tr><td>1</td></tr>
<tr><td>2</td></tr>
<tr><td>3</td></tr>
<tr><td>4</td><td rowspan="4">当前时间</td></tr>
<tr><td>5</td></tr>
<tr><td>6</td></tr>
<tr><td>7</td></tr>
<tr><td>8</td><td rowspan="4">时间运行范围</td></tr>
<tr><td>9</td></tr>
<tr><td>10</td></tr>
<tr><td>11</td></tr>
<tr><td>12</td><td rowspan="4">保留 - 报警时间</td></tr>
<tr><td>13</td></tr>
<tr><td>14</td></tr>
<tr><td>15</td></tr>
<tr><td>16</td><td rowspan="6">保留 - 运行小时</td></tr>
<tr><td>17</td></tr>
<tr><td>18</td></tr>
<tr><td>19</td></tr>
<tr><td>20</td></tr>
<tr><td>21</td></tr>
<tr><td>22</td><td rowspan="5">运行 / 停止后的
时间 / 日期</td></tr>
<tr><td>23</td></tr>
<tr><td>24</td></tr>
<tr><td>25</td></tr>
<tr><td>26</td></tr>
</table>

表 19

时钟数据区中的当前时间：

DW	字/左	字/右
4	---	星期几
6	日	月
7	年	AM/PM (位，编号 7)，小时
8	分	秒

图 5

时钟数据区中的设置区：

DW	字/左	字/右
9	闰年	星期几
10	日	月
11	年	AM/PM (位，编号 7)，小时
12	分	秒

图 6

5.2.11.2　时钟数据区 CPU 928B、CPU 948

对数据字编号的规定是相对的。区域的实际位置由调用参数：UDAT = FB 86: MESS:CLOCK 的 DB 编号、DW 编号来确定。

DW	位置
0	内部变量
1	
2	
3	
4	当前时间
5	
6	
7	
8	时间运行范围
9	
10	
11	

图 7

时钟数据区中的当前时间：

DW	字/左								字/右							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
4	秒								0							
5	格式		小时						分							
6	月中的第几天								星期几				0			
7	年								秒							

图 8

设置区中的当前时间：

DW	字/左								字/右							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
8	秒								0							
9	格式		小时						分							
10	月中的第几天								星期几				0			

图 9

5.2.11.3 时钟数据区 CPU 946、CPU 947

对数据字编号的规定是相对的。区域的实际位置由调用参数：UDAT = FB 86:MESS:CLOCK 的 DB 编号、DW 编号来确定。

DW	位置
0	内部变量
1	
2	
3	
4	当前时间
5	
6	
7	
8	时间运行范围
9	
10	
11	

图 10

时钟数据区域的当前时间：

DW	字/左		字/右	
4	10 秒	1 秒	1/10 秒	1/100 秒
6	10 小时	1 小时	10 分钟	1 分钟
7	10 天	1 天	星期几	0
8	10 年	1 年	10 个月	1 个月

图 11

设置区域的当前时间：

DW	字/左		字/右	
9	10 秒	1 秒	1/10 秒	1/100 秒
10	10 小时	1 小时	10 分钟	1 分钟
11	10 天	1 天	星期几	0
12	10 年	1 年	10 个月	1 个月

图 12

5.2.11.4 消息块的时钟数据格式

对数据字编号的规定是相对的。区域的实际位置由参数 ZCLOCK = FB 86: MESS:CLOCK 的 DB 编号、DW 编号来确定。

如果按时间顺序的消息采集功能和 S5 报警系统一起使用，则需要将数据 DB 80、DW 190 输入到参数 ZCLOCK 中。

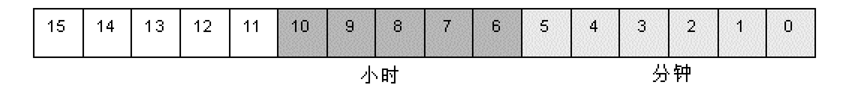
功能块 FB 86: MESS:CLOCK 使日期和时间可以二进制代码形式用于消息处理：

设置区域的当前时间：

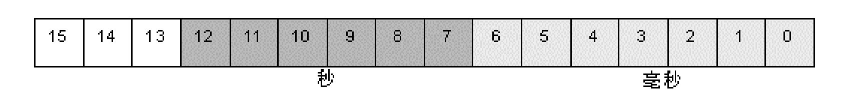
描述	数据字	位	有效范围	注释
1/100 秒	1	0 - 6	0 至 99 (0 - 990 毫秒)	以一个 10 毫秒的光栅为单位
秒	1	7 - 12	0 至 59	
分	0	0 - 5	0 至 59	
小时	0	6 - 10	0 至 23	
日	2	0 - 4	1 至 31	
月	2	5 - 8	1 至 12	
年	2	9 - 15	0 至 127 (1990-2117)	年 + 1990

图 13

DW3：时间



DW4：时间



DW4：日期

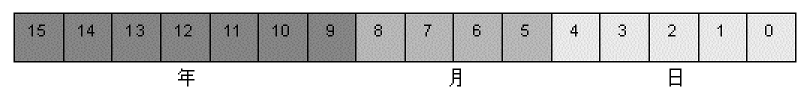


图 14

5.2.12 接口说明

使用 S5 时间同步软件，用户需要：

- 填写 FB 86: MESS:CLOCK 的调用参数一章中所描述的 FB 86: MESS:CLOCK 的调用参数
- 打开 PLC 中的数据区

组态实例

假定使用具有两个 PG 接口的 CPU 944。在该 CPU 上，将为不带 DCF77 时钟的 S5 报警系统建立 S5 时间同步。
数据区：

时间同步消息：	DB 100、DW 20 - DW 30
S5 系统时钟的时钟数据区	DB 100、DW 31 - DW 47
消息块数据*：	DB 80、DW 190 - DW 192
*	SIMATIC S5 报警系统的应用程序需要使用该数据区。

在系统中组态通道参数(例如工业以太网)时，必须为时间同步条目输入所期望的数据块(DB 100、DW20 - DW 30)。
必须确保从 DW 0 至 DW 255 的 DB 80 和从 DW 0 至 DW 47 的 DB 100 是打开的。

从 OB 1 引用：

```
: SPA   FB 86
NAME  : MELD: UHR
CPUT  : KF +1
DCF7  : KF +0
QTYP  : KF +0
QSYN  : KY 100, 20
UDAT  : KY 100, 31
ZINT  : KF +0
ZUHR  : KY 80, 190
ZSYN  : KF +0
```

5.2.13 与 WinCC 报警系统的相互作用

以下有关 WinCC 报警系统与 S5 消息块的相互作用必须加以考虑：
在 S5 的发送块中，必须将要传送的数据字数目指定为 256。
在控制中心内，必须为 S5 传输通道单元建立新驱动程序连接。在连接标签中，指定读功能为被动取。
为了与报警系统进行数据交换，必须为每个 PLC 创建两个原始数据变量。
第一个变量负责接收消息。
按如下设置其编址。数据区：DB；DB 号：xx；地址：字；DW：0；原始数据类型：事件
第二个变量负责发送确认信息。
按如下设置其编址。数据区：DB；DB 号：80；地址：字；DW：90；原始数据类型：事件
在报警系统中，将事件变量与接收原始数据变量相连(在这种情况下位信息无意义)。
将确认变量与发送原始数据变量相连(在这种情况下位信息无意义)。
输入文件 S5STD.NLL 作为格式 DLL。
提示：通过互连向导，可以在一个操作中连接所有相关的消息。
作为过程值，只有正的定点数才有效。不支持浮点数。

S5	WinCC
过程值 ⇨	过程值 1
作业描述 ⇨	过程值 2
成批数据描述 ⇨	过程值 3
保留 ⇨	过程值 4

5.3 与报警记录和变量记录的格式 DLL 接口

目的

报警记录和变量记录应用程序通过 WinCC 数据管理器获得过程数据。根据过程的通讯类型，

- 数据传送涉及不同的通道 DLL，
- 过程数据存储在不同结构的消息(原始数据变量)中。

然而，无论使用何种通讯类型，报警记录和变量记录都将以同样的方法处理过程数据。因此，要为每种通讯类型使用独立的格式 DLL，它了解有关消息的确切结构，并可从中获取用于报警记录和变量记录的通用的过程数据形式。

格式 DLL 本质上属于通道 DLL，它象通道 DLL 那样可以很容易地添加到整个系统和从整个系统中删除。尽管如此，它与相关的通道 DLL 没有直接接口。

本文描述了报警记录和变量记录 WinCC 应用程序的每个格式 DLL 的集成和接口。由于该文本是在设计 S7PMC 格式 DLL 时生成的，所以术语 S7PMC 格式 DLL 与术语格式 DLL 基本同义。

基本过程

S7PMC 格式 DLL 是一个被动的程序组，它仅具有与报警记录和变量记录应用程序的接口。S7PMC 格式 DLL 处理用于报警记录和变量记录的 S7PMC 指定的函数。

通过使用开始调用，可将报警记录和变量记录登录到格式 DLL 上。在这种情况下，特定的参数在开始结构中传送到格式 DLL 并通过 ID 接收其属性。

在对运行系统中的 S7PMC 函数进行处理时，需要两个数据传送方向：

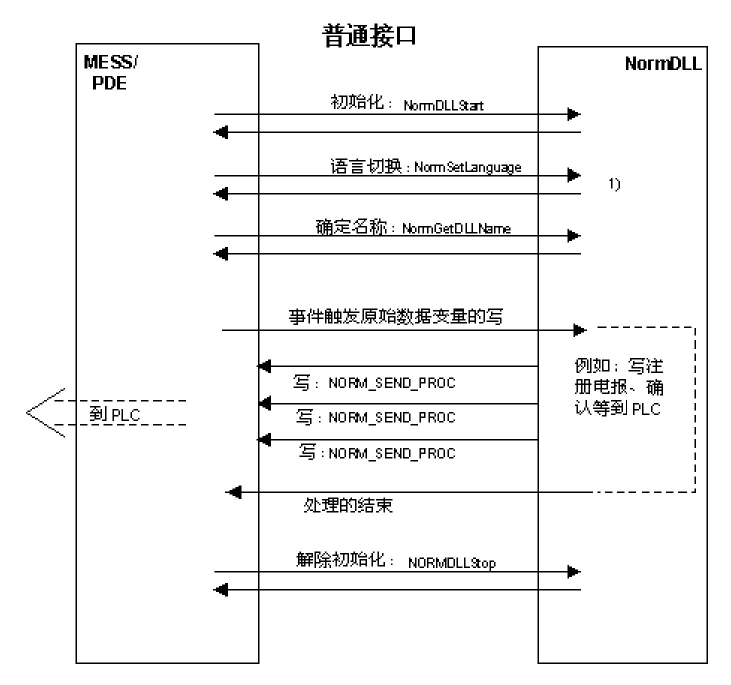
- OS 到 PLC：(登录/退出作业、确认的发送)
- PLC 到 OS：(消息和归档数据的接收)

通过初始化调用，变量记录/报警记录将所组态的归档变量名称和消息号传送给 S7PMC DLL。对于这些对象，格式 DLL (WinCC)必须登录到 PLC 上。可以在任何时候对初始化调用进行处理。

为在取消初始化时返回资源等，由报警记录/变量记录调用格式 DLL。

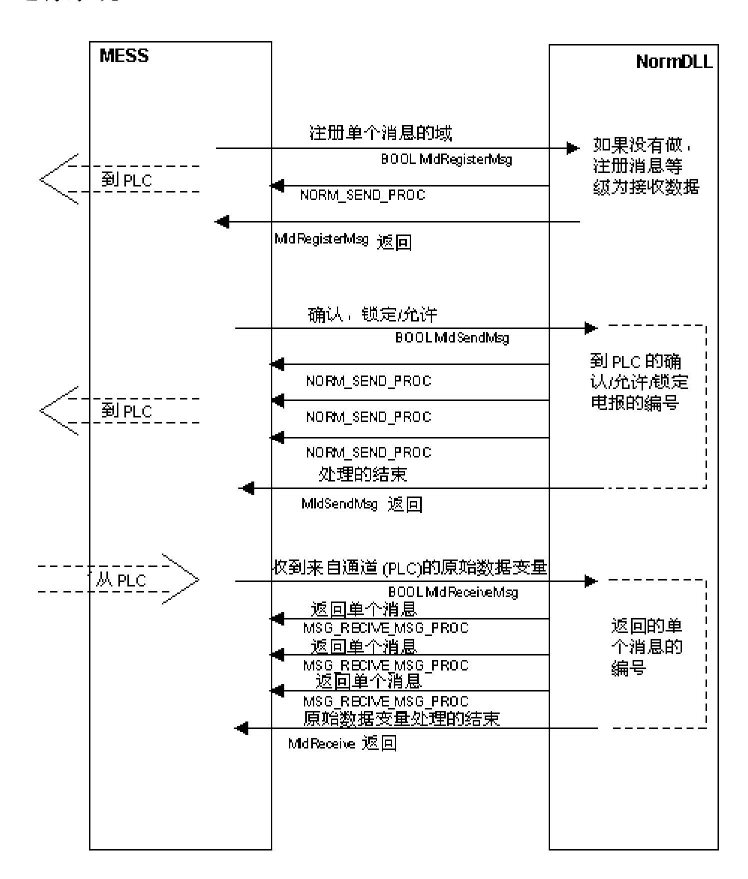
5.3.1 与报警记录和变量记录的共享接口

对报警记录和变量记录完全相同的格式 DLL 的常规函数，被编组在一个共享的接口中。函数名称全部以 NORM...开头。
(报警记录指定函数的前缀为：Mld...，变量记录指定函数的前缀为：Pde...。)



MELD = 报警记录
PDE = 变量记录

对指定报警记录的补充
运行系统

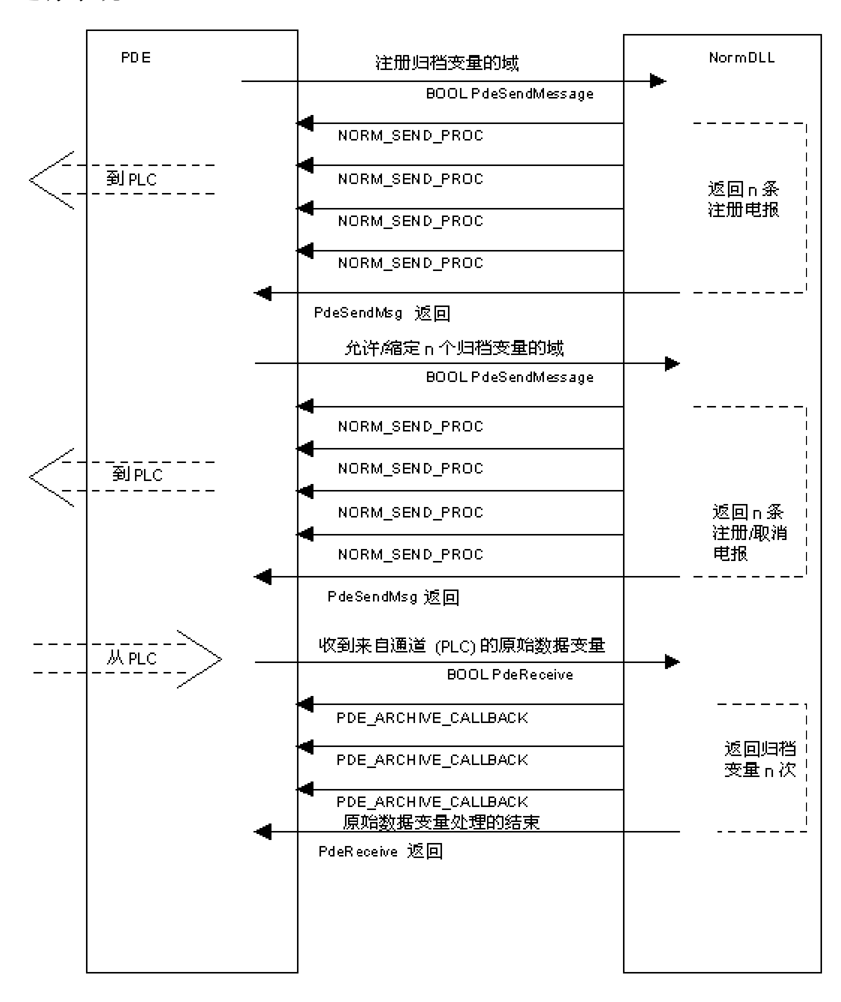


扩展的组态对话框

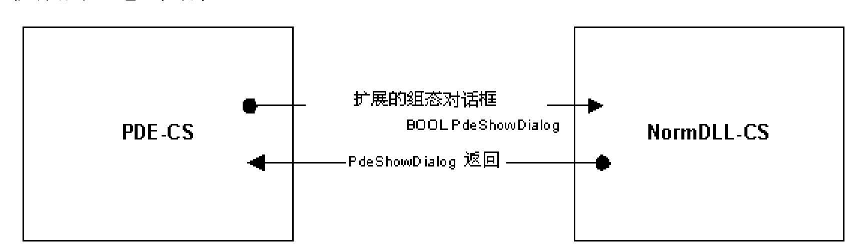


5.3.2 对指定变量记录的补充

运行系统



扩展的组态对话框



5.3.3 WinCC 格式 DLL 的 API 函数

格式 DLL 划分成下列子区域：

- 格式 DLL 的初始化
- 当装载格式 DLL 时(LibMain)，由操作系统进行初始化
- 格式 DLL 的属性的查询
- 格式 DLL 的名称的查询
- 关闭格式 DLL
- 由变量记录和报警记录关闭
- 由操作系统卸载
- 对组态的扩展
- 消息组态期间的对话框扩展
- 归档变量组态期间的对话框扩展
- 在线服务
- 所有格式 DLL 指定对象(消息、归档变量)的注册
- 语言切换
- 格式化
- 消息的格式化
- 归档变量的格式化

5.3.3.1 格式 DLL 的初始化

装载操作期间进行初始化

报警记录和/或变量记录应用程序借助 LoadLibrary 系统调用函数来装载 WinCC 格式 DLL。随后，由操作系统装载格式 DLL 并通过其标准机制进行初始化。定义格式 DLL 的全部条目地址。

5.3.3.2 格式 DLL 的属性的查询

报警记录与变量记录通过调用 NormDLLStart 登录到相关的格式 DLL。这样做可使格式 DLL 和应用程序间进行信息交换。
NormDLLStart

```
#include <winccnrm.h>

BOOL NormDLLStart(
        LPVOID lpUser,
        BOOL bModeRuntime,
        PNORM_STARTSTRUCT pcis,
        PCMN_ERROR lpError);
```

参数	描述
lpUser	指向应用数据的指针，不加改变地转送至回调函数
bModeRuntime	如果格式 DLL 在运行模式下启动则为 TRUE，如果它在组态模式下启动则为 FALSE；当前它未由格式 DLL 进行判断
pcis	指向开始结构的指针
lpError	指向缺省 WinCC 错误结构的指针

返回值	描述
TRUE	没有错误
FALSE	API 函数出错，通过指针 lpError 描述出错原因

NORM_STARTSTRUCT

组件	描述	I/O
dwSize	结构的大小(按字节计)	O
lpstrProjectPath	当前所选项目的路径	I
NORM_SEND_PROC pfnWriteRwData	指向应用程序的回调函数的指针，通过它格式 DLL 可以借助数据管理器将原始数据变量发送给 PLC。	I
dwAppID	应用程序 ID： 1 = 报警记录 2 = 变量记录 3 = 用户(为将来的应用程序而保留，当前未使用)	I
dwLocalID	调用时的语言设置	I
dwNormCap	格式 DLL 的属性参见下表	O

用于将原始数据变量发送至 WinCC 数据管理器的回调函数提供如下：

```
typedef BOOL(*NORM_SEND_PROC)(
        LPDM_VAR_UPDATE_STRUCT    lpDmVarUpdate,
        DWORD                     dwWait,
        LPVOID                    LpUser,
        LPCMN_ERROR               lpError );
```

参数	描述
lpDmVarUpdate	指向原始数据变量的指针
dwWait	有关应用程序是否应该等到完成写调用的标识： 带 SET_VALUE 的 WAIT_ID_NO 带 SET_VALUE_WAIT 的 WAIT_ID_YES
lpUser	指向应用数据的指针，调用 NormDLLStart 时被通知
lpError	指向缺省 WinCC 错误结构的指针

返回值	描述
TRUE	没有错误
FALSE	API 函数出错，通过指针 lpError 描述出错原因

根据此表为每种属性分配一个位：

定义	位屏蔽		描述
NORMCAP_DIALOG	0x00000001	设置	格式 DLL 提供一个特殊的对话框
		已删除	格式 DLL 不提供对话框
NORMCAP_REENTRANT	0x00000002	设置	格式 DLL 可以重入
		已删除	格式 DLL 不可以重入
NORMCAP_MSG_FREE_LOCK	0x00000004	设置	消息可以登录/退出
		已删除	消息不可以登录/退出
NORMCAP_ARC_FREE_LOCK	0x00000008	设置	归档变量可以登录/退出
		已删除	归档变量不可以登录/退出
NORMCAP_MSG_GENERIC	0x00000010	设置	消息可以按类别生成
		已删除	消息不可以按类别生成
NORMCAP_ARC_GENERIC	0x00000020	设置	归档变量可以按类别生成
		已删除	归档变量不可以按类别生成

5.3.3.3 格式 DLL 的名称的查询

NormGetDLLName

```
#include <winccnrm.h>
LPTSTR NormGetDLLName( void );
```

返回值	描述
LPTSTR	指向一个字符串的指针，该字符串包含以简洁语言描述的格式 DLL 的名称；名称取决于当前的语言设置。

5.3.4 关闭格式 DLL

由变量记录和报警记录关闭

变量记录和报警记录通知格式 DLL：应用程序是否关闭。然后，在格式 DLL 中正确返回资源。
NormDLLStop

```
#include <winccnrm.h>

LPTSTR NormGetDLLName( void );
```

返回值	描述
TRUE	函数应用成功
FALSE	API 函数出错

由操作系统卸载

不必采用特殊的预防措施。

5.3.4.1 对组态的扩充

S7PMC 对象需要某些规定。在对话框中首先使用标准方法(没有 MFC)请求这些规定，然后直接加入 WinCC 消息号或归档变量名称。也就是说，格式 DLL 不必自己存储和管理这些规定。为了保证消息号或归档变量在整个项目中的唯一性，需要在消息号或归档变量和相关的原始数据变量之间进行分配。该分配信息是消息号或归档变量名称中的一个完整部分。

5.3.4.2 组态 S7PMC 消息时的对话框扩充

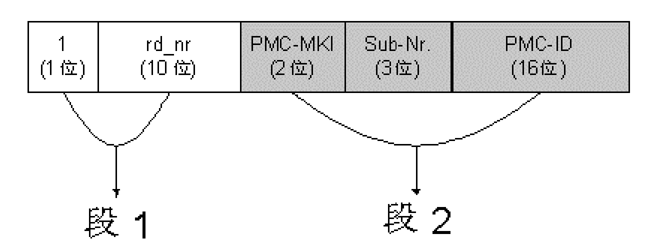
格式 DLL 具有用于定义 S7PMC 指定消息号的 API 函数。当将参数分配给属于 S7PMC 格式 DLL 的单个消息时，由 CS 报警系统调用该函数。由 S7PMC 格式 DLL 分配的消息号由两部分组成。

第 1 部分：

唯一标识项目范围内 PLC CPU 的编号(原始数据变量编号)。

第 2 部分：

属于来自 PLC 侧消息的编号，唯一标识 PLC CPU (格式 DLL 指定)中的消息。
在组态对话框中，必须完成下列选择以便建立消息号：
S7PMC 消息号的结构(32 位)



关于第 1 部分

每个消息属于标识 PLC CPU 的原始数据变量。为了将原始数据变量分配给消息号，已经建立了下列定义。
S7PMC 原始数据变量的名称(以及所有与格式 DLL 的连接类型)具有以下固定结构：
@rd_alarm#rd_nr

@rd_alarm#	格式 DLL 的原始数据变量名称的固定组成部分
rd_nr	用于标识原始数据变量的 0 与 1023 之间的十进制数(无先行零)

为由格式 DLL 分配的消息号(外部)设置消息号的最高有效位。这些消息只能由相应的格式 DLL 进行处理，也就是说通过报警记录的组态对话框不能更改消息号。

关于第 2 部分

消息号的这部分可以由相应的格式 DLL 进行分配。对于 S7PMC，含义如下：

MKI	消息级别；必须选择下列级别之一： SCAN (1) ALARM/NOTIFY (2) ALARM_8P/ALARM_8 (2) LTM (3)
子号	子消息号只适用于 ALARM_8 和 ALARM_8P： 1...8
PMC-ID	PMC 消息号(块输入参数 EV-ID)： 1...16386 用于消息级别 SCAN 与 ALARM/NOTIFY 或者 ALARM_8P/ALARM_8 1...7 用于消息级别 LTM

MldShowDialog

```
#include <winccnrm.h>

BOOL WINAPI MldShowDialog(
             HWND                    hwnd,
             LPMSG_CSDATA_GENERIC    lpmCS,
             LPDM_PROJECT_INFO       lpDMProjectInfo,
             LPCMN_ERROR             lpError );
```

参数	描述
hwnd	窗口句柄
LpmCS	指向单个消息数据的指针
LpDMProjectInfo	指向项目信息结构的指针
LpError	指向缺省 WinCC 错误结构的指针

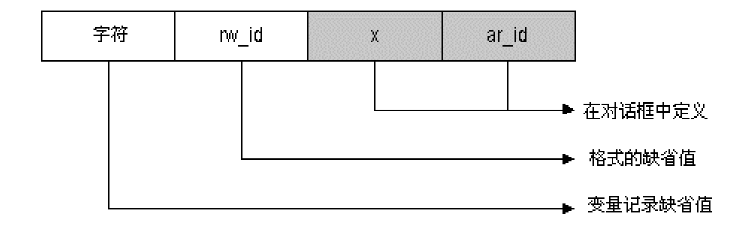
返回值	描述
TRUE	函数应用成功
FALSE	API 函数出错，通过指针 lpError 描述出错原因

5.3.4.3 在组态归档变量时对话框的扩充

格式 DLL 有一个 API 函数，用于定义 S7PMC 指定的归档变量名称。在将参数分配到属于 S7PMC 连接的归档变量时，由 CS 变量记录调用该函数。由 S7PMC 格式 DLL 分配的归档变量名称由几部分组成，其中包含了属于 PLC 归档的编号。用此规则，在 WinCC 归档变量描述中唯一包含了 S7PMC 变量编号，这样就能够在运行时进行最快分配。

变量记录确保归档变量名称全都是唯一的。

S7PMC 归档变量名称的结构(不大于 18 个字节)



名称	字节长度	由...分配	描述
字符	9	变量记录	由变量记录分配的固定字符串，包括格式 DLL 名称和分隔符#，例如对 S7PMC: NRMS7PMC，不通过界面显示
rw_id	8	变量记录/ 格式 DLL	采用十六进制字符格式(包括先行零)的原始数据 ID，可对包括了归档编号的原始数据变量(连接)进行唯一分配。名称部分由格式 DLL 使用变量记录输入参数形成。
x	1	格式 DLL CS 部分	S7PMC 指定的 ID 用于区分 BSEND 和 AR_SEND： A = AR_SEND B = BSEND
ar_id	4	格式 DLL CS 部分	采用十六进制字符格式的 ID (包括先行零) 根据 x ID： S7PMC 指定的归档编号 AR_ID 或 在 BSEND 时 S7 指定的 R_ID

为 S7PMC 所生成的归档变量名称的实例：#00000001#A#0014
PdeShowDialog

```
#include <winccnrm.h>

BOOL WINAPI PdeShowDialog(
                LPVOID          hwnd,
                LPTSTR          lpszArcVarName,
                DWORD           dwArcVarNameLength,
                LPDM_VARKEY     lpVarKey,
                LPCMN_ERROR     lpError
                );
```

参数	描述
hwnd	窗口句柄
lpszArcVarName	指向字符串域的指针，该域存储了格式 DLL 指定的归档变量名称部分
dwArcVarNameLength	格式 DLL 指定的名称部分的最大长度
lpVarKey	指向原始数据变量 Varkey 的指针
lpError	指向 WinCC 错误结构的指针
返回值	**描述**
TRUE	函数应用成功
FALSE	API 函数出错，通过指针 lpError 对出错原因进行描述

5.3.4.4 在线服务

注册所有消息

由于格式 DLL 没有相关消息的组态信息，因而需要该函数。但在应用程序(WinCC)为接收消息而登录之前，PLC 不会发送消息。报警记录当前为每个相关消息调用 MldRegisterMsg 函数，并以此方法将单个消息的组态信息传送给格式 DLL。除了消息描述以外，格式 DLL 还接收一个指向分配给该消息的原始数据变量的指针(连接)。也就是说，格式 DLL 能在运行系统的主存储器中创建一个表，通过该表可对 S7PMC 指定的登录消息进行构造。
MldRegisterMsg

```
#include <winccnrm.h>

BOOL WINAPI MldRegisterMsg(
                LPDM_VARKEY     lpDMVarKey,
                LPDWORD         lpMsgNumber,
                DWORD           DwNumMsgNumber,
                LPCMN_ERROR     lpError );
```

参数	描述
lpDMVarKey	指向原始数据变量 Varkey 的指针
lpMsgNumber	指向具有单个消息号的域的指针
dwNumMsgNumber	单个消息号的数量
lpError	指向 WinCC 错误结构的指针
返回值	描述
TRUE	函数应用成功
FALSE	API 函数出错，通过指针 lpError 对出错原因进行描述

5.3.4.5 注册所有归档变量

由于格式 DLL 没有相关归档变量的组态信息，因而需要用到该函数。因此要为一些相关的归档变量调用 PdeSendMsg 函数。这样可使归档变量了解变量记录的组态信息和其它信息。
一个连接的多个归档变量可通过单个调用来注册。
对于每个归档变量，变量记录都将一个双字作为附加信息传送至格式 DLL，其保留在格式 DLL 存储器的存储区域中。一旦必须处理归档变量时，变量记录就需要该附加信息(在回调函数 TagLogging_ARCHIVE_CALLBACK 中)。
也就是说，运行期间格式 DLL 能在主存储器中创建一个表，通过该表可为各归档构造 S7PMC 指定的登录消息。登录消息要用于向 PLC 声明已准备接收各归档编号的状态。仅在成功登录后，PLC 才会将归档数据发送到应用程序(WinCC)。
PdeSendMsg

```
#include <winccnrm.h>

BOOL WINAPI PdeSendMsg(
        NORM_SEND_PROC      lpfnCallBack,
        DWORD               dwFunctionId,
        LPSZ_ARC_VAR_NAME   lpszArcVarName,
        LPDWORD             lpdwData
        DWORD               dwNumArchVarName,
        LPDM_VARKEY         lpVarKey,
        LPVOID              lpUser,
        LPCMN_ERROR         lpError
        );
```

参数	描述
lpfnCallBack	指向回调例程的指针，通过该例程可把由格式 DLL 生成的原始数据变量传送给数据管理器(DM)。 如果为零，则从 INI 结构中调用回调例程。 INI 结构中的函数地址不同于此参数。
dwFunctionId	函数标识符 FUNC_ID_REGISTER (与下面的表格相比较)，相同的函数适用于所有列出的变量
lpszArcVarName	指向一个指针域的指针，该指针域的元素为归档变量名称
lpdwData	指向一个域的指针，该域的元素包含归档变量的附加数据，也可以为零。 通过 FUNC_ID_REGISTER 函数(登录归档变量)，将归档变量的附加值不作修改地传送至格式 DLL 的内部列表，并在合适的时侯转送至 TagLogging_ARCHIVE_CALLBACK。 对其它函数标识符没有意义。
dwNumArchVarName	要处理的归档变量名称的数量
lpVarKey	指向原始数据变量 Varkey 的指针
lpUser	指向用户数据的指针，不作修改即传送给回调函数
lpError	指向 WinCC 错误结构的指针
返回值	**描述**
TRUE	函数应用成功
FALSE	API 函数出错，通过指针 lpError 描述出错原因

PdeSendMsg 过程可能具有的功能(来自 dwFunctionId 的数值)：

定义	位屏蔽	描述
FUNC_ID_LOCK	0x00000001	锁定归档变量
FUNC_ID_FREE	0x00000002	激活归档变量
FUNC_ID_REGISTER	0x00000004	注册归档变量
FUNC_ID_UNREGISTER	0x00000008	取消注册归档变量(当前不需要)

5.3.4.6 语言切换

组态对话框必须依赖于语言，即格式 DLL 必须识别当前设置的语言。语言设置可在启动时在起始结构中报告。动态语言切换也必须由变量记录和报警记录转送给格式 DLL。为此，可使用以下调用：

NormSetLanguage

```
#include <winccnrm.h>

BOOL NormSetLanguage(
              DWORD           dwLocaleID,
              LPCMN_ERROR     lpError
              );
```

参数	描述
dwLocalID	调用时的语言设置
lpError	指向缺省 WinCC 错误结构的指针
返回值	描述
TRUE	函数应用成功
FALSE	API 函数出错，通过指针 lpError 对出错原因进行描述

5.3.5 格式化

如果应用程序已经在 PLC 上登录以用于接收消息或归档数据，则将通过各原始数据变量提供该数据。在注册时登录格式 DLL。从那时起，可以由 PLC 发送数据消息。数据消息包括在原始数据变量中，并通过通道 DLL、数据管理器和有关的应用程序传送给格式 DLL (例如变量记录或报警记录)；格式 DLL 负责原始数据变量类型。格式 DLL 翻译进入的数据并构造消息或从中归档数据。

5.3.5.1 单个消息的获取

原始数据变量(消息)的内容可以存储 n 条单个消息。格式 DLL 必须解释此 S7PMC 指定的消息，并将得到的单个消息转发至报警记录。

S7PMC 的消息号(EV_ID)是 WinCC 消息号的一部分。

在一条单个消息中，至多可以由 S7PMC 传递 10 个过程值。在这种情况下，允许过程值为字符串类型。报警记录不支持这种过程值类型，因此格式 DLL 必须拒绝这种类型的附加数值。

每次原始数据变量的状态改变(即如果数据管理器在正常状态之后检测到故障状态或相反)时，由报警记录调用 MldReceiveMsg 函数。原始数据变量(相当于连接)的状态改变只是对 S7PMC 格式 DLL 比较重要。附加信息可以参见状态改变时的处理一章。

MldReceiveMsg

```
#include <winccnrm.h>

BOOL WINAPI MldReceiveMsg(
               MSG_RECEIVE_MSG_PROC    lpfnMsgReceive,
               LPDM_VAR_UPDATE_STRUCT  lpDMVar,
               LPVOID                  lpUser,
               LPCMN_ERROR             lpError );
```

参数	描述
lpfnMsgReceive	指向回调例程的指针，通过该例程把由格式 DLL 构造的单个消息传送至报警记录。
lpDMVar	指向原始数据变量的指针
lpUser	指向用户数据的指针，不作修改即传送给回调函数
lpError	指向 WinCC 错误结构的指针
返回值	描述
TRUE	函数应用成功
FALSE	API 函数出错，通过指针 lpError 描述出错原因

用于将单个消息发送至报警记录的回调函数提供如下：

```
typedef BOOL (*MSG_RECEIVE_MSG_PROC)(
               LPMSG_RTCREATE_STRUCT   lpMsgCreate,
               DWORD                   dwNumMsg,
               LPVOID                  lpUser,
               LPCMN_ERROR             lpError);
```

参数	描述
lpMsgCreate	指向 WinCC 消息的指针
dwNumMsg	单个消息的数量
lpUser	指向应用数据的指针
lpError	指向缺省 WinCC 错误结构的指针
返回值	**描述**
TRUE	函数应用成功
FALSE	API 函数出错，通过指针 lpError 描述出错原因

5.3.5.2 确认、锁定/激活消息

WinCC 报警记录的消息和报警概念以及 S7PMC 为根据消息组态来确认消息做好准备。确认信息为报警记录所知，但还必须在 PLC 的消息确认存储区中对其进行管理。为此，报警记录使用与连接相关的格式 DLL 将确认消息发送给 PLC。

根据输入的数据，S7PMC 格式 DLL 构造相应的 S7PMC 消息，并由报警记录回调函数 NORM_SEND_PROC 将该消息转送给数据管理器。

如果要由报警记录锁定/再次激活单个消息(即根据 PLC 中的源数据禁止/允许消息的生成)，则采用相同的过程。

MldSendMsg

```
#include <winccnrm.h>

BOOL WINAPI MldSendMsg(
              NORM_SEND_PROC            lpfnMsgSend,
              LPMSG_SEND_DATA_STRUCT    lpSendData,
              DWORD                     dwNumData,
              LPVOID                    lpUser,
              LPCMN_ERROR               lpError );
```

参数	描述
lpfnMsgSend	指向报警记录回调例程的指针，使用该例程传送由格式 DLL 构造的原始数据变量，以写入 PLC。参数描述在格式 DLL 的属性的查询一章。
lpSendData	指向发送数据的指针，其结构详细描述如下
dwNumData	要处理的各个作业的数量
lpUser	指向用户数据的指针，不作修改即传送给回调函数
lpError	指向 WinCC 错误结构的指针
返回	**描述**
TRUE	函数应用成功
FALSE	API 函数出错，通过指针 lpError 描述出错原因

报警记录发送数据(单个作业)的结构

变量	描述
DWORD dwVarID	DM 的原始数据变量 ID
DWORD dwNotify	注意：可能值 MSG_STATE_QUIT：确认消息 MSG_STATE_LOCK：锁定消息 MSG_STATE_UNLOCK：激活消息 MSG_STATE_QUIT_EMERGENCY：确认所有消息
DWORD dwData	QUIT、LOCK、UNLOCK --> 消息号 EMERGENCY ACK --> 未使用

5.3.5.3 状态改变时的处理

连接(原始数据变量)的状态改变必须向格式 DLL 报告。这由 MldReceiveMsg 函数来完成。

状态改变	S7PMC 格式 DLL 中的处理
故障 - 正常	将所有 S7PMC 消息级别的登录消息(至少已组态了一条消息)传送给 PLC。为指定的 S7PMC 消息级别进行登录。 格式 DLL 已知道所有的组态消息，因为消息进行了注册。
正常 - 故障	格式 DLL 必须拒绝那些已被发送到 PLC 但由于状态改变(确认丢失)而不能被完全处理的激活作业。

5.3.5.4 S7PMC 格式 DLL 的消息更新

在消息更新时，S7PMC 格式 DLL 读取已向它报告的所有消息的消息状态(通过注册)，并将其作为单个消息发送给报警记录。因此，在系统启动时可以基于一致的消息画面。

在以下情况下需要消息更新：

- 检测到状态从故障变为正常(系统启动也是这种情况)
- PLC 发送消息更新的消息。如果检测到消息溢出，则将该消息发送到每个已登录的参与者，来自其它参与者的消息将被确认或激活。

消息更新期间，PLC 发送消息确认状态和锁定 ID。不发送附加值和时间。在这种情况下，格式 DLL 提供当前系统时间作为单个消息的时间，或将 ID MSG_STATE_UPDATE 写入消息状态。

5.3.5.5 归档变量的格式化

格式 DLL 为变量记录提供两个功能：

- 从原始数据变量的内容获取各归档变量数值
- 锁定/激活归档变量

5.3.5.6 各归档变量值的获取

原始数据变量(消息)的内容可以存储 n 个归档变量值。格式 DLL 必须解释此 S7PMC 指定的消息，并将得到的归档变量值转送给变量记录。

对于归档变量，也可以发送过程值转换程序。然后，S7PMC 格式 DLL 将执行从过程值到归档变量值的转换。此过程涉及现有的 WinCC 定标函数。准确的步骤仍需指定。

每次原始数据变量的状态改变(即如果数据管理器在正常状态之后检测到故障状态或相反)时，由变量记录调用 PdeReceive 函数。原始数据变量(相当于连接)的状态改变只是对 S7PMC 格式 DLL 比较重要。附加信息可以参见状态改变时的处理一章。

PdeReceive

```
#include <winccnrm.h>

BOOL PdeReceive (
            LPDM_VAR_UPDATE_STRUCT        lpDmVarUpdate,
            TagLogging_ARCHIVE_CALLBACK   lpfnCallBack,
            LPVOID                        lpUser,
            LPCMN_ERROR                   lpError
            );
```

参数	描述
lpDmVarUpdate	指向原始数据变量的指针
lpfnCallBack	指向回调例程的指针，通过该例程格式 DLL 将各归档变量值传送给变量记录。
lpUser	指向用户数据的指针，不作修改即传送给回调函数
lpError	指向 WinCC 错误结构的指针
返回值	**描述**
TRUE	函数应用成功
FALSE	API 函数出错，通过指针 lpError 描述出错原因

提供了如下的回调函数，用于将各归档变量值发送给变量记录：

```
BOOL (*PDE_ARCHIVE_CALLBACK) (
            LPTSTR         lpszArcVarName,
            double         doValue,
            SYSTEMTIME*    lpstTime
            DWORD          dwFlags
            DWORD          dwData,
            LPVOID         lpUser,
            LPCMN_ERROR    lpError
            );
```

参数	描述
lpszArcVarName	与原始数据 ID 中相同的归档变量名称
doValue	归档变量值
lpstTime	指向时间标志的指针，该标志从原始数据变量的用户数据中获取
dwFlags	ID，其确切的含义仍需指定。
dwData	注册时提供的附加日期，传送时无需修改
lpUser	指向用户数据的指针，从调用函数中应用它时不加修改
lpError	指向 WinCC 错误结构的指针
返回值	**描述**
TRUE	函数应用成功
FALSE	API 函数出错，通过指针 lpError 描述出错原因

5.3.5.7 锁定/激活归档变量

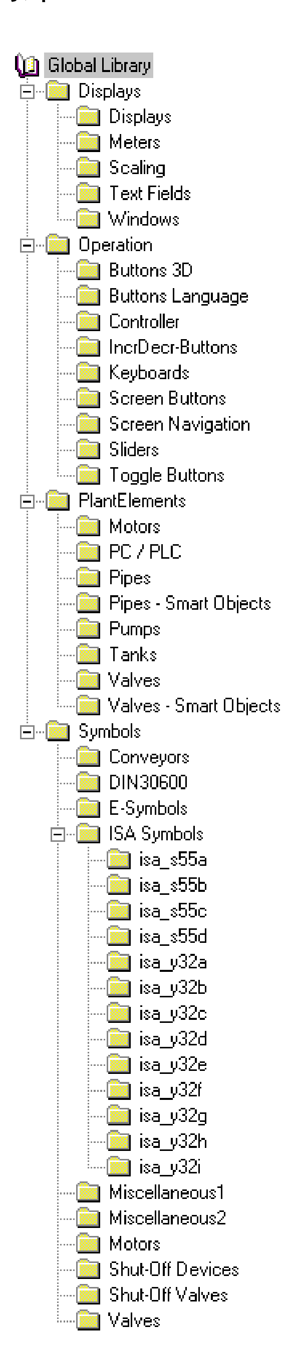
使用该函数，变量记录可在 S7PMC 中控制归档变量数值的接收。为此，S7PMC 格式 DLL 建立了用于退出或登录相应归档的调用函数，并将该调用函数通过 NORM_SEND_PROC 传送给 DM。
从 S7PMC 格式 DLL 来看，归档变量的锁定/激活几乎与注册归档变量所需的函数完全相同。因此对于这两种函数，都以格式 DLL 调用同样的函数(PdeSendMsg)。
通过 dwFunctionId 函数标识符，对注册和锁定/激活加以区别：对于锁定/激活，用于每个归档变量的附加数据 lpdwData 没有意义。参见注册所有归档变量一章。

5.3.5.8 状态改变时的处理

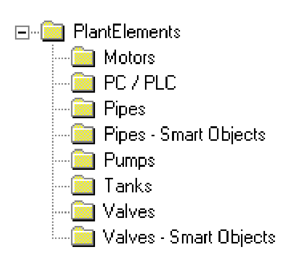
连接(原始数据变量)的状态改变必须向格式 DLL 报告。这由 PdeReceive 函数来完成。

状态改变	S7PMC 格式 DLL 中的处理
故障 - 正常	所有连接的全部归档变量的登录消息。 格式 DLL 已知道所有的组态消息，因为消息进行了注册。
正常 - 故障	格式 DLL 必须拒绝接收那些已发送到 PLC 但由于状态改变(确认丢失)而不能被完全处理的激活作业。

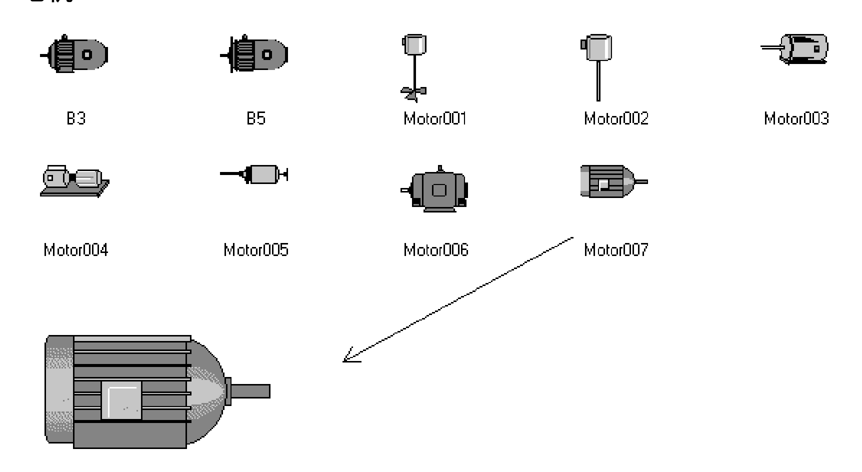
5.4 全局库


Global Library
Displays
Displays
Meters
Scaling
Text Fields
Windows
Operation
Buttons 3D
Buttons Language
Controller
IncrDecr-Buttons
Keyboards
Screen Buttons
Screen Navigation
Sliders
Toggle Buttons
PlantElements
Motors
PC / PLC
Pipes
Pipes - Smart Objects
Pumps
Tanks
Valves
Valves - Smart Objects
Symbols
Conveyors
DIN30600
E-Symbols
ISA Symbols
isa_s55a
isa_s55b
isa_s55c
isa_s55d
isa_y32a
isa_y32b
isa_y32c
isa_y32d
isa_y32e
isa_y32f
isa_y32g
isa_y32h
isa_y32i
Miscellaneous1
Miscellaneous2
Motors
Shut-Off Devices
Shut-Off Valves
Valves

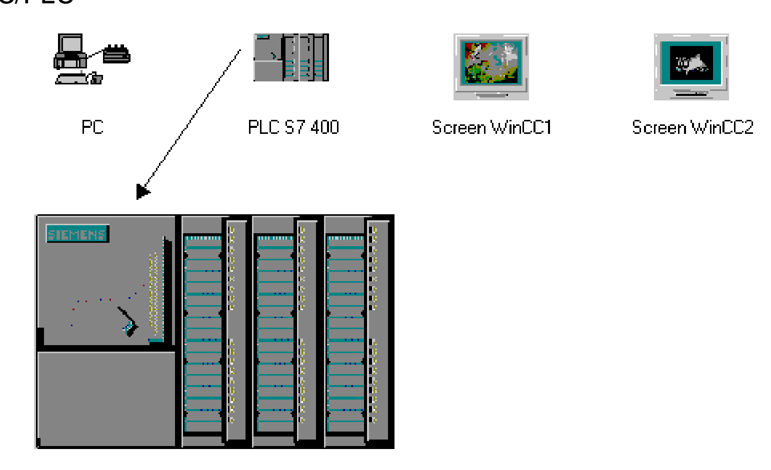
5.4.1 系统块



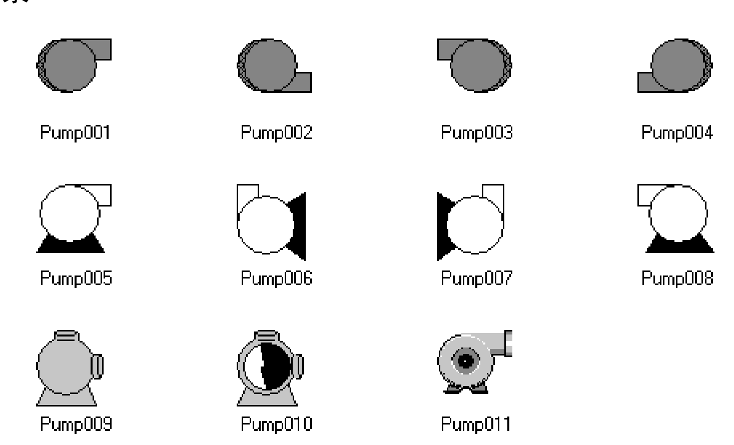
5.4.1.1 电机



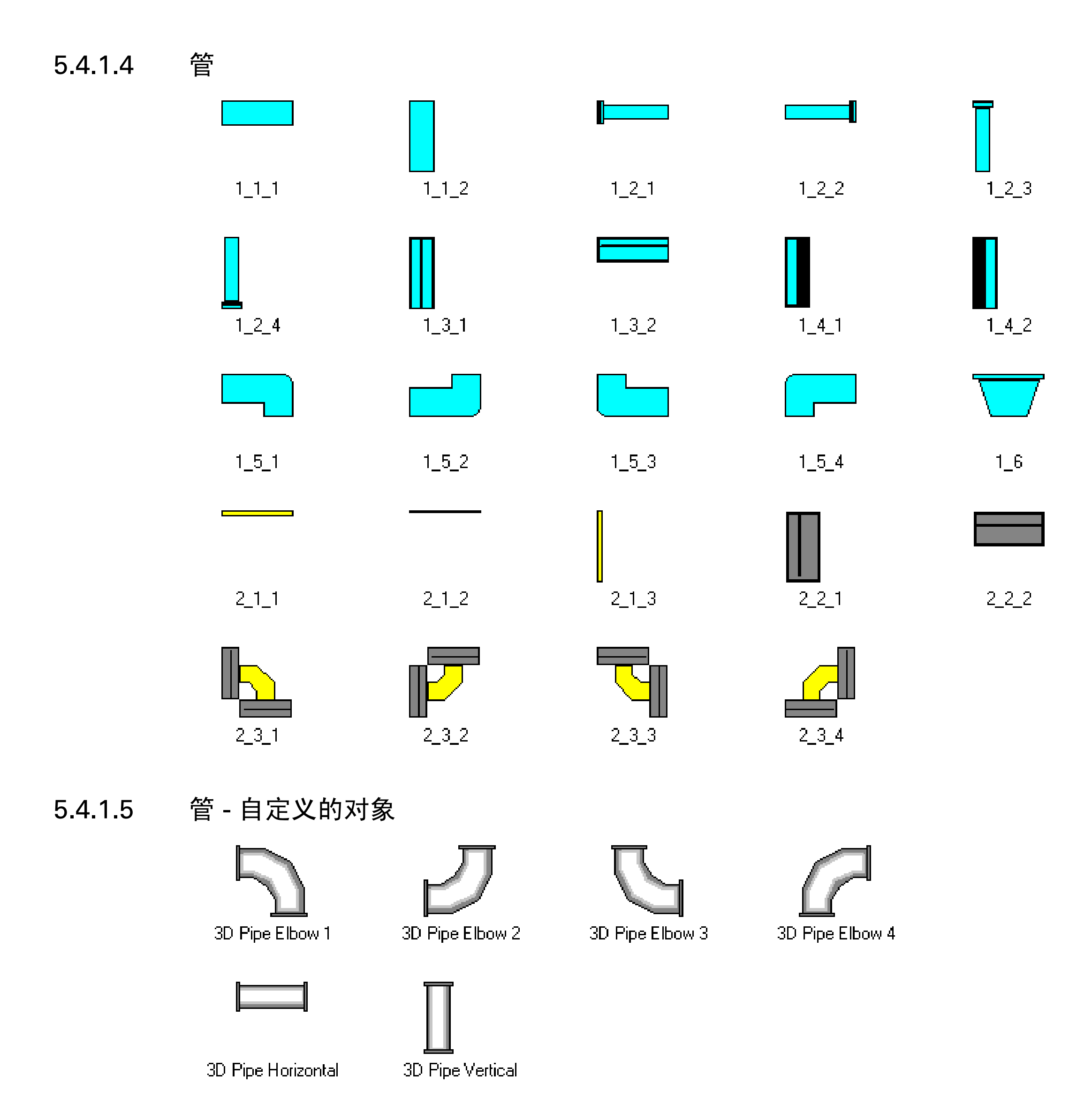
5.4.1.2 PC/PLC



5.4.1.3 泵

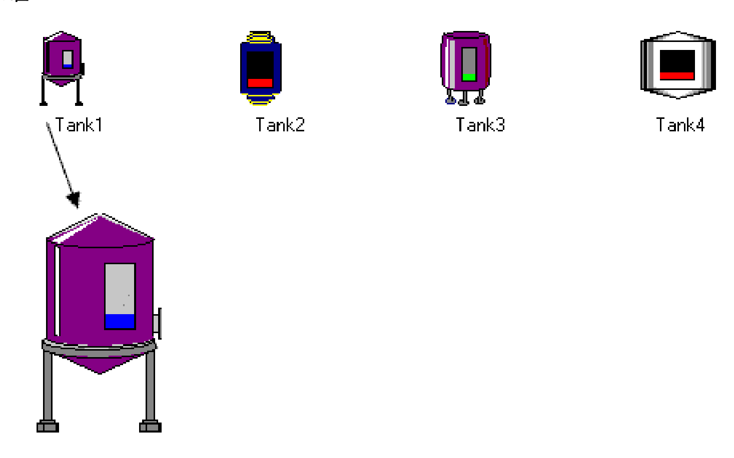


5.4.1.4 管

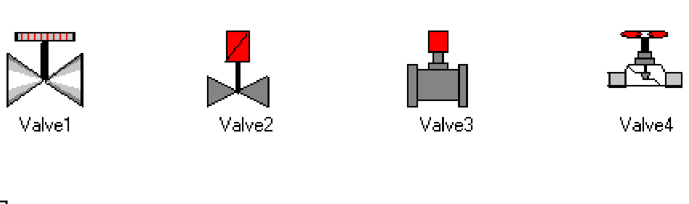


5.4.1.5 管 - 自定义的对象

5.4.1.6 罐

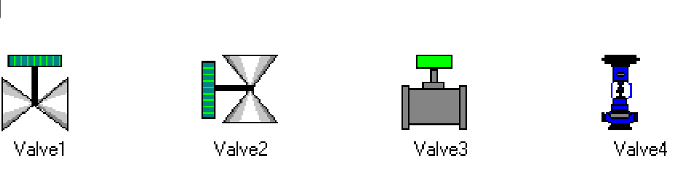


5.4.1.7 阀 - 自定义的对象

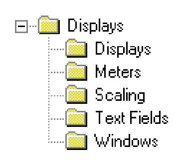




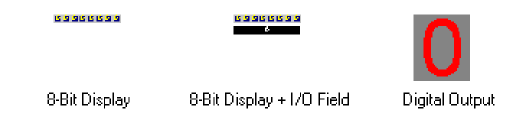
5.4.1.8 阀



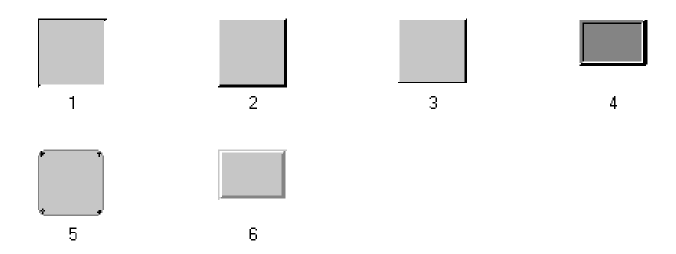
5.4.2 显示



5.4.2.1 显示



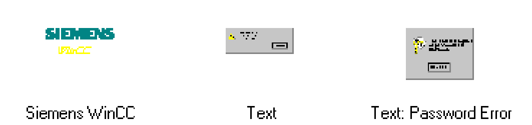
5.4.2.2 窗口



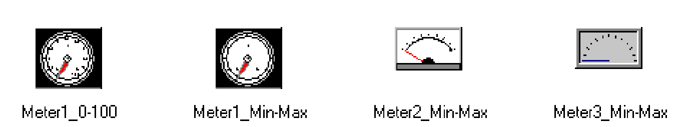
5.4.2.3 定标度



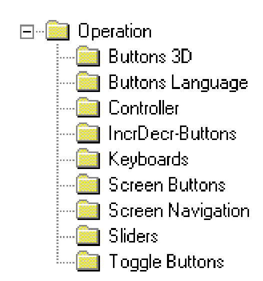
5.4.2.4 文本域



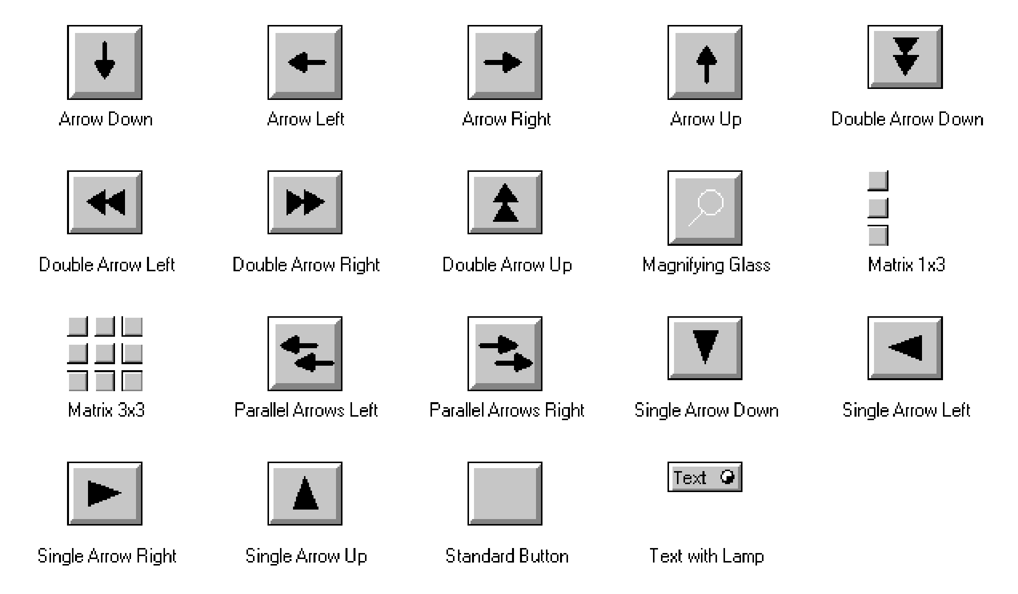
5.4.2.5 仪表



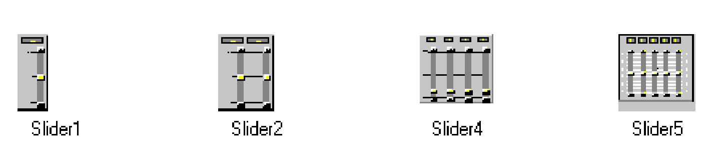
5.4.3 控件



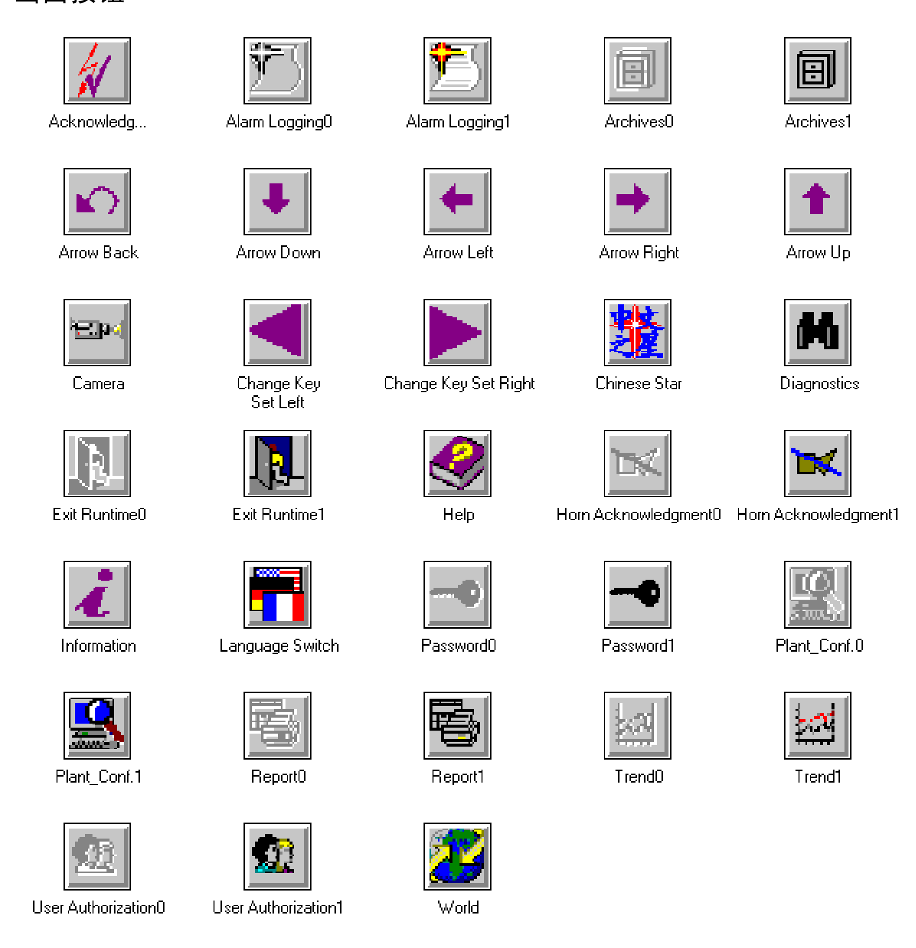
5.4.3.1 3D 按钮



5.4.3.2 控制面板



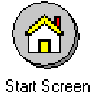
5.4.3.3 画面按钮



5.4.3.4 画面浏览



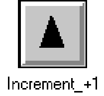


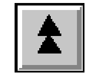



5.4.3.5 递增/递减按钮

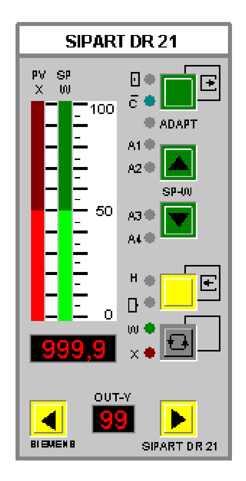




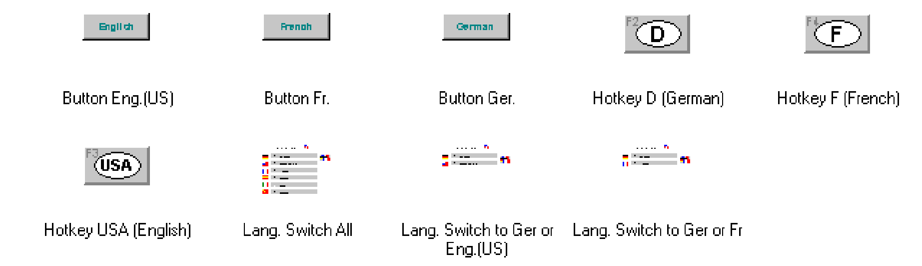


5.4.3.6 控制器



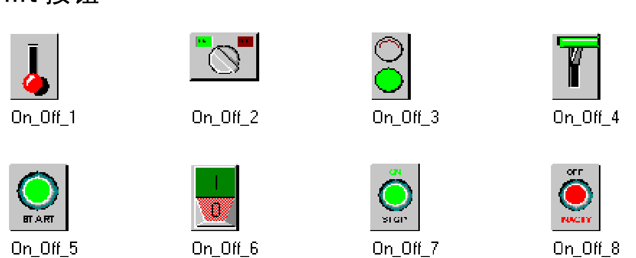
5.4.3.7 语言切换



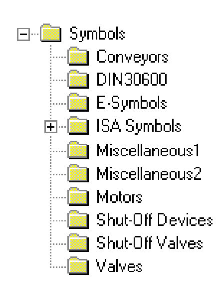
5.4.3.8 键盘



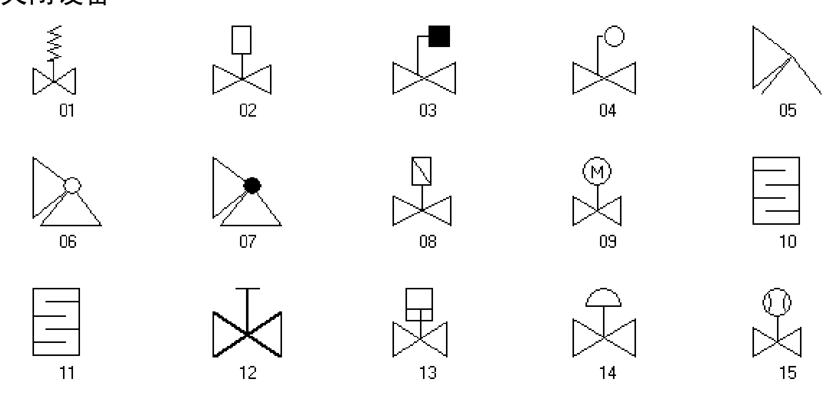
5.4.3.9 Shift 按钮



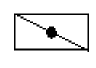
5.4.4 符号

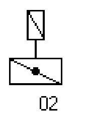


5.4.4.1 关闭设备

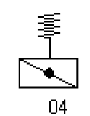


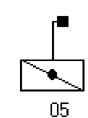
5.4.4.2　关闭阀

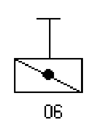

01

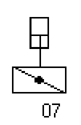

02

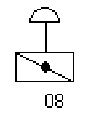
03

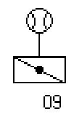

04

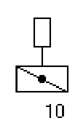

05

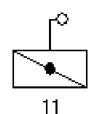

06

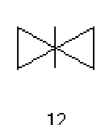

07

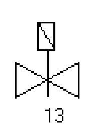

08

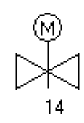

09


10

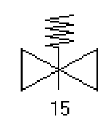

11

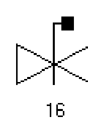

12

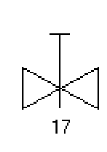

13

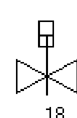


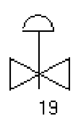
14

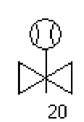

15

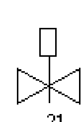

16


17


18

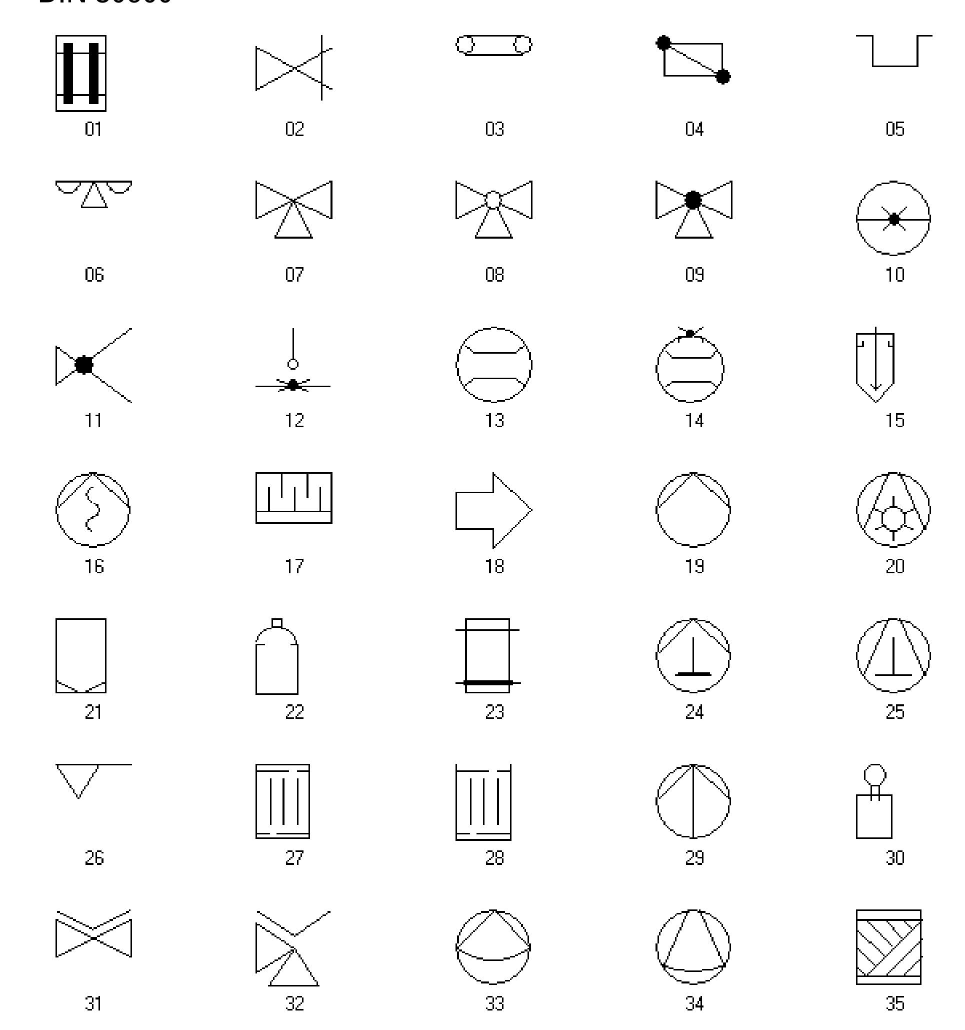

19


20

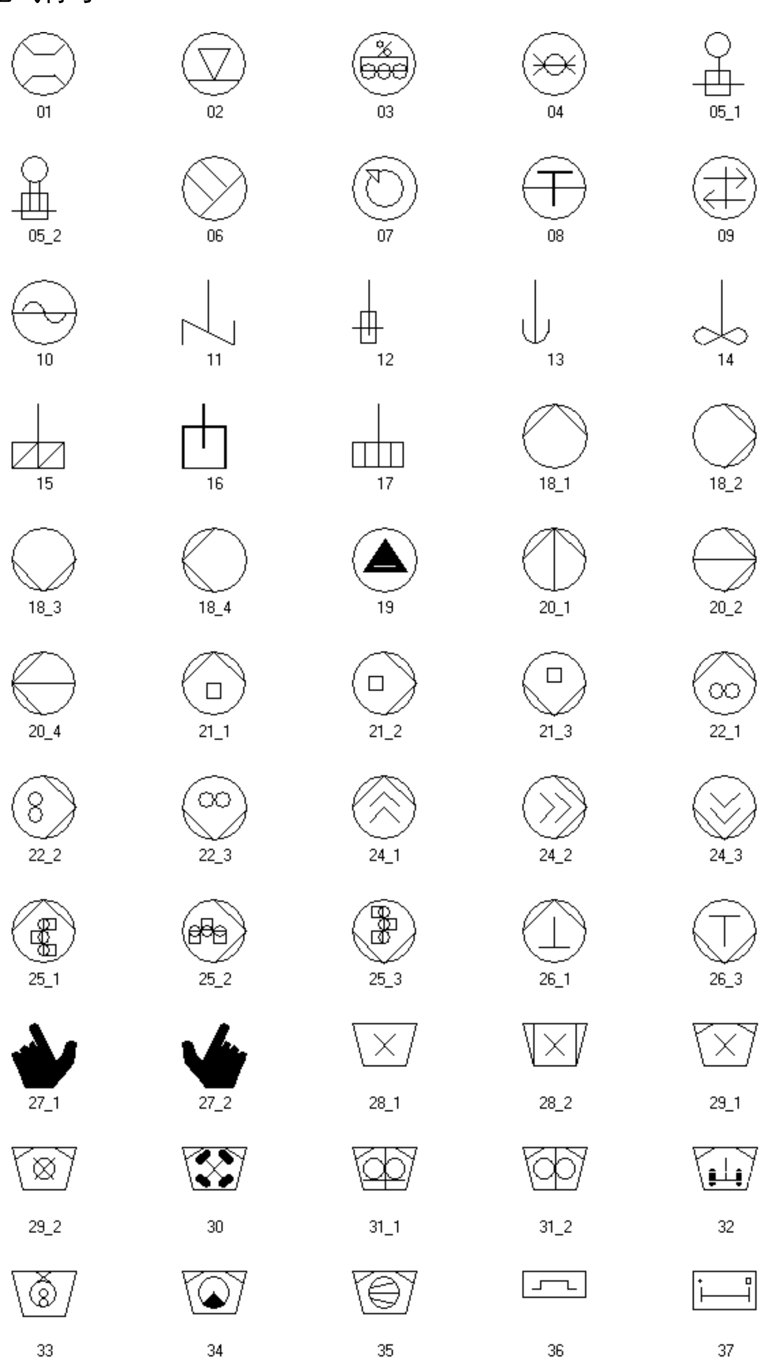

21

22

5.4.4.3 DIN 30600



5.4.4.4 电气符号


01
02
03
04
05_1
05_2
06
07
08
09
10
11
12
13
14
15
16
17
18_1
18_2
18_3
18_4
19
20_1
20_2
20_4
21_1
21_2
21_3
22_1
22_2
22_3
24_1
24_2
24_3
25_1
25_2
25_3
26_1
26_3
27_1
27_2
28_1
28_2
29_1
29_2
30
31_1
31_2
32
33
34
35
36
37

5.4.4.5 传送带

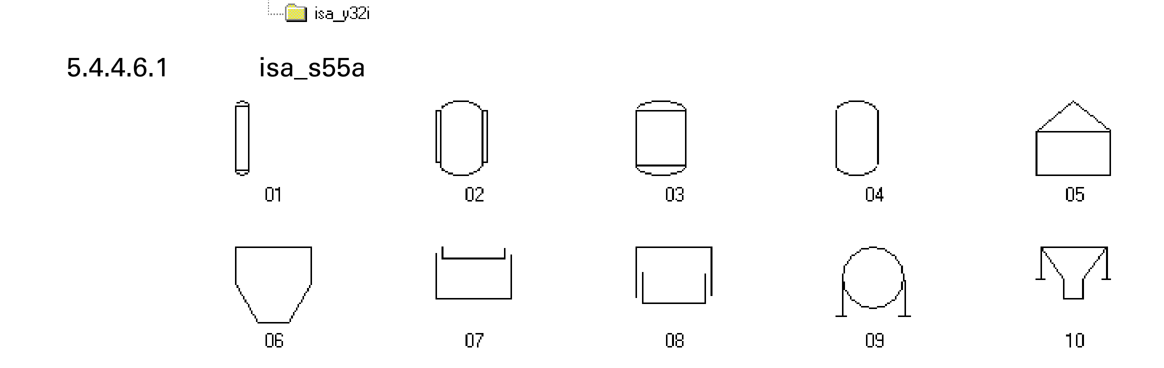
1 2 3

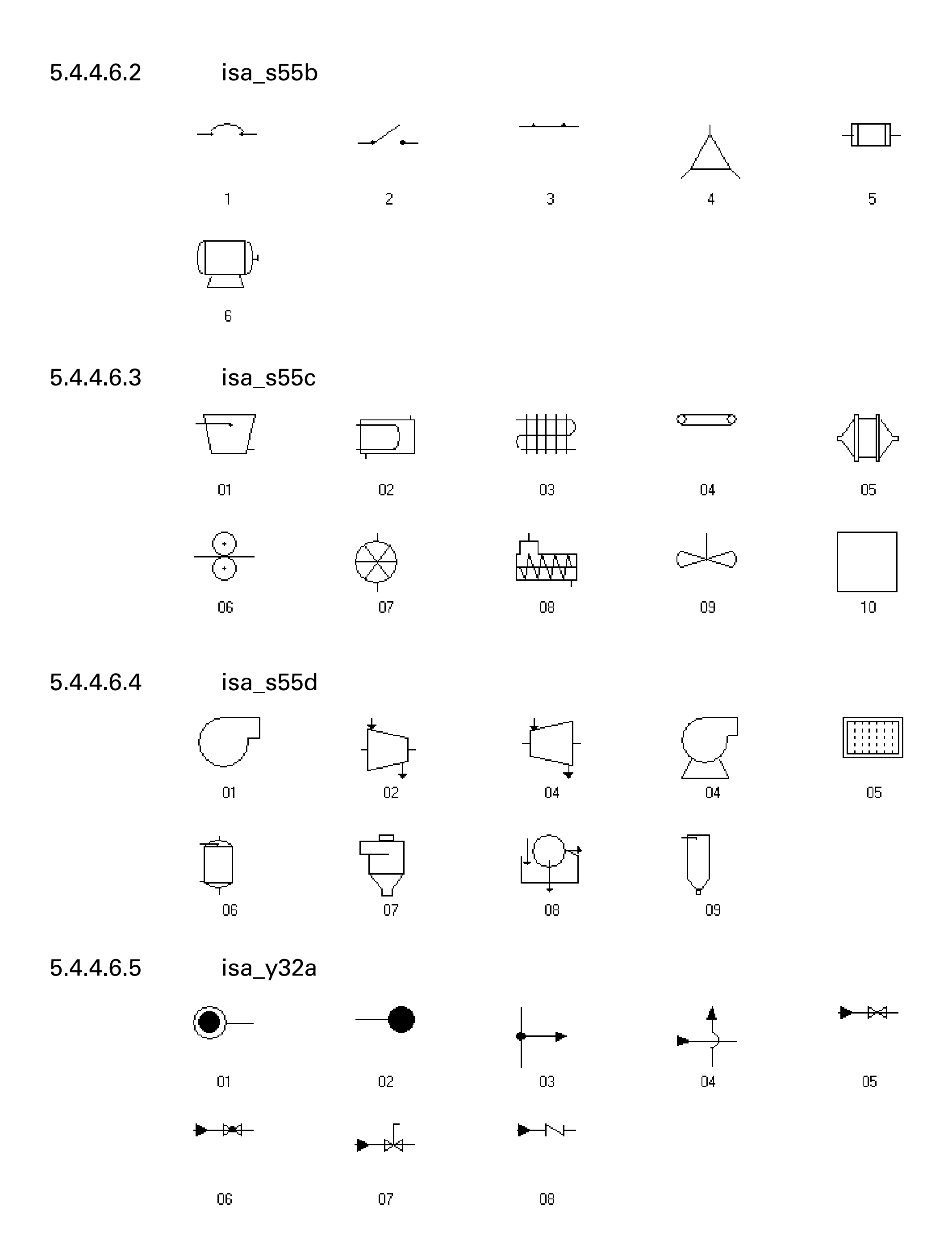
4 5 6

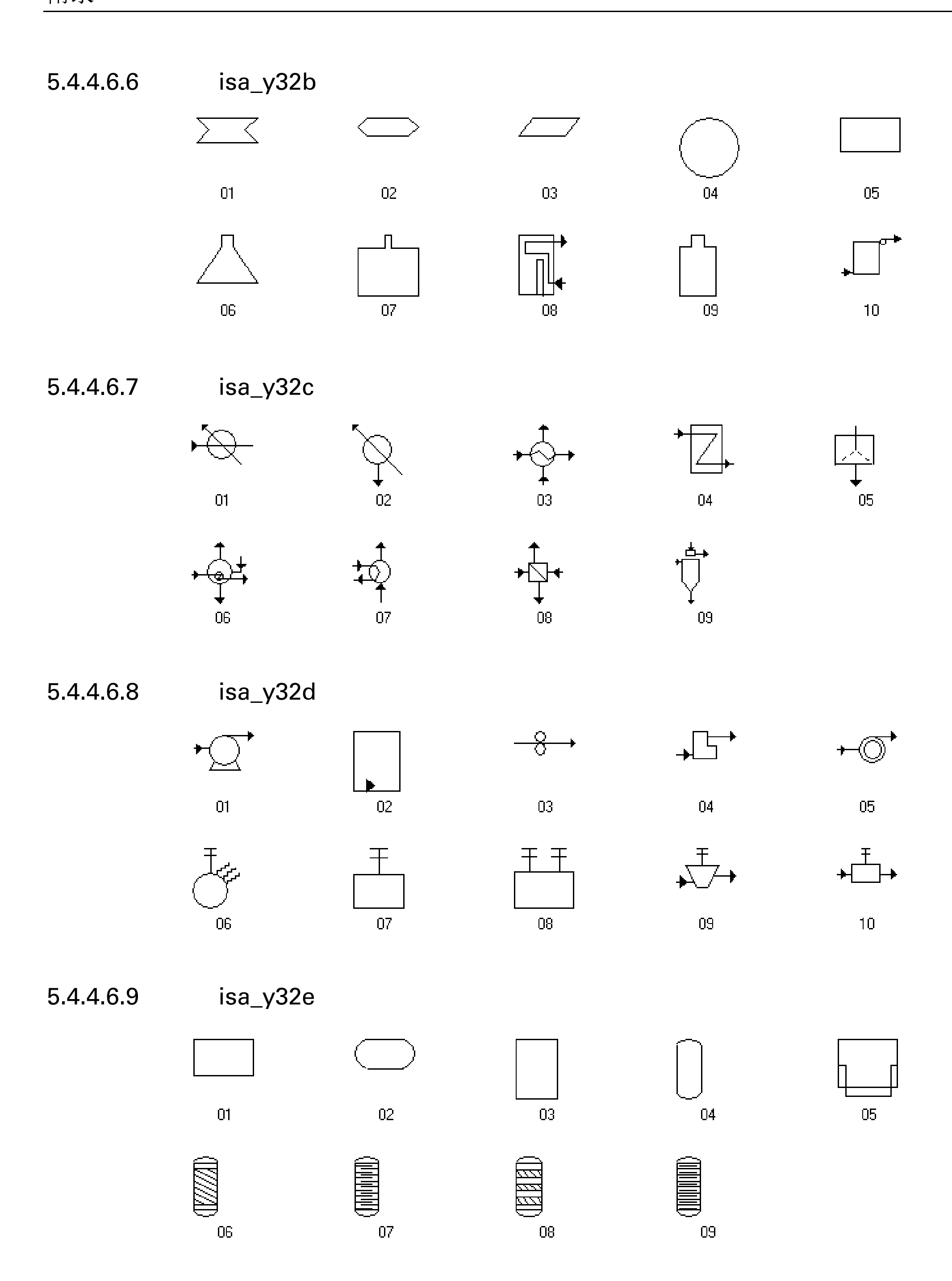
5.4.4.6 ISA 符号

- ISA Symbols
  - isa_s55a
  - isa_s55b
  - isa_s55c
  - isa_s55d
  - isa_y32a
  - isa_y32b
  - isa_y32c
  - isa_y32d
  - isa_y32e
  - isa_y32f
  - isa_y32g
  - isa_y32h
  - isa_y32i

5.4.4.6.1 isa_s55a




5.4.4.6.2 isa_s55b
1
2
3
4
5
6
5.4.4.6.3 isa_s55c
01
02
03
04
05
06
07
08
09
10
5.4.4.6.4 isa_s55d
01
02
04
04
05
06
07
08
09
5.4.4.6.5 isa_y32a
01
02
03
04
05
06
07
08



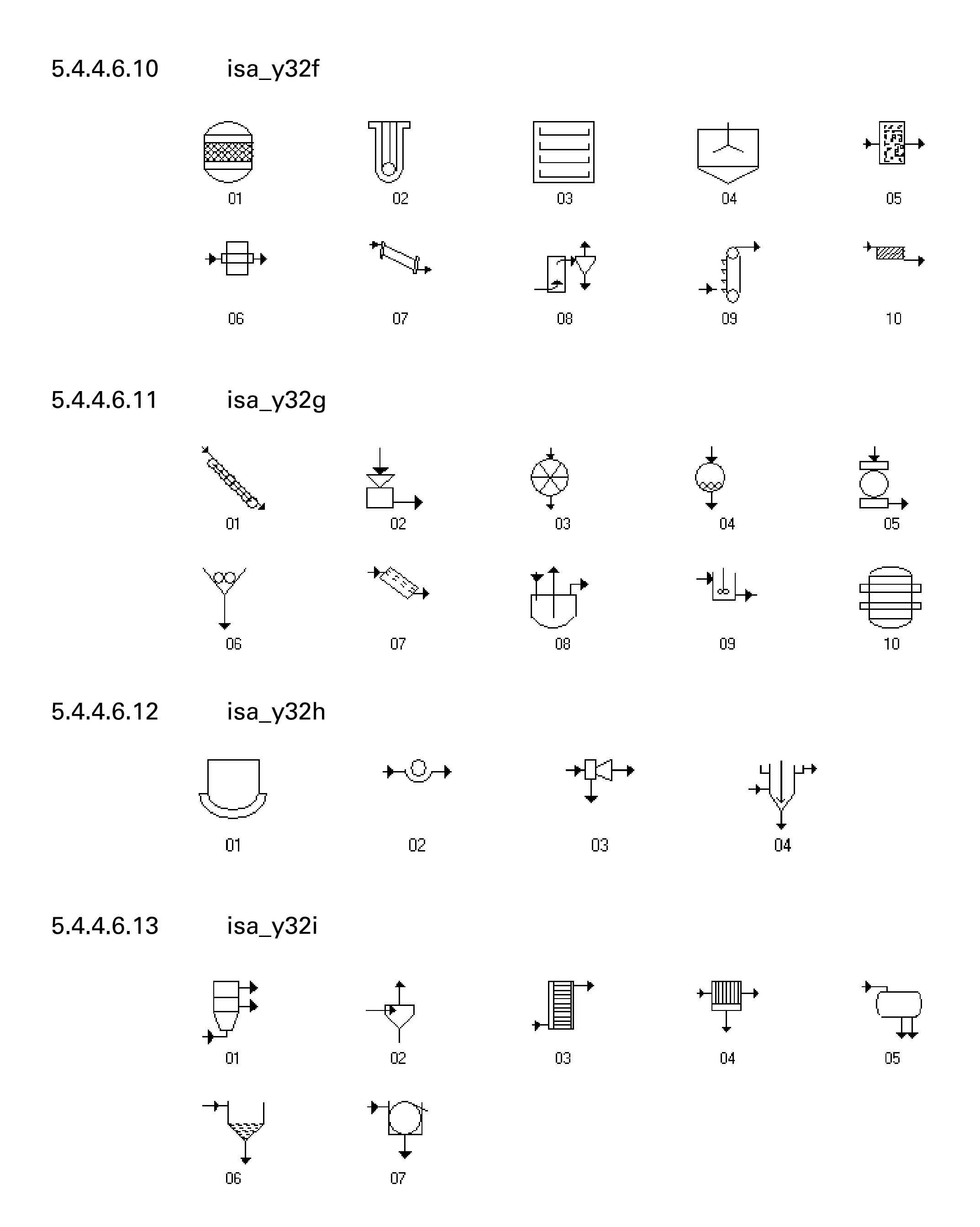
5.4.4.6.6 isa_y32b
01 02 03 04 05
06 07 08 09 10
5.4.4.6.7 isa_y32c
01 02 03 04 05
06 07 08 09
5.4.4.6.8 isa_y32d
01 02 03 04 05
06 07 08 09 10
5.4.4.6.9 isa_y32e
01 02 03 04 05
06 07 08 09

5.4.4.6.10 isa_y32f

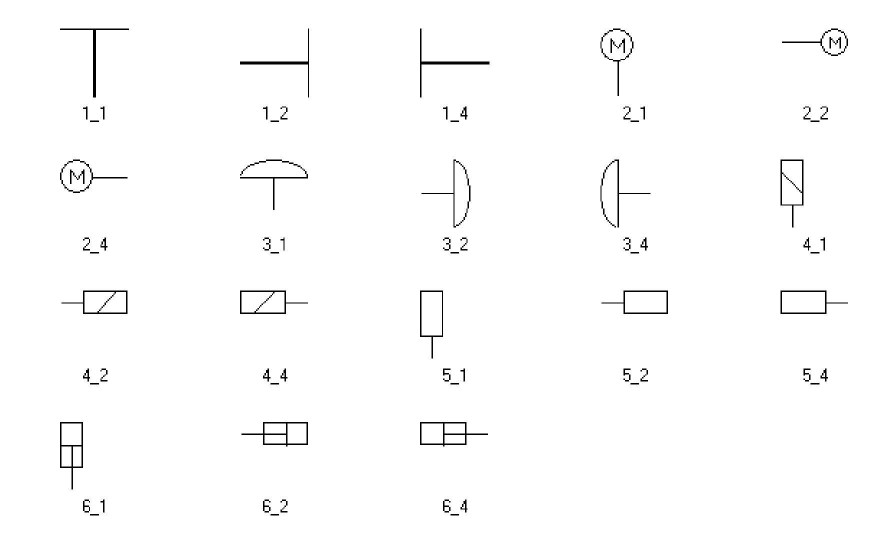
5.4.4.6.11 isa_y32g

5.4.4.6.12 isa_y32h

5.4.4.6.13 isa_y32i

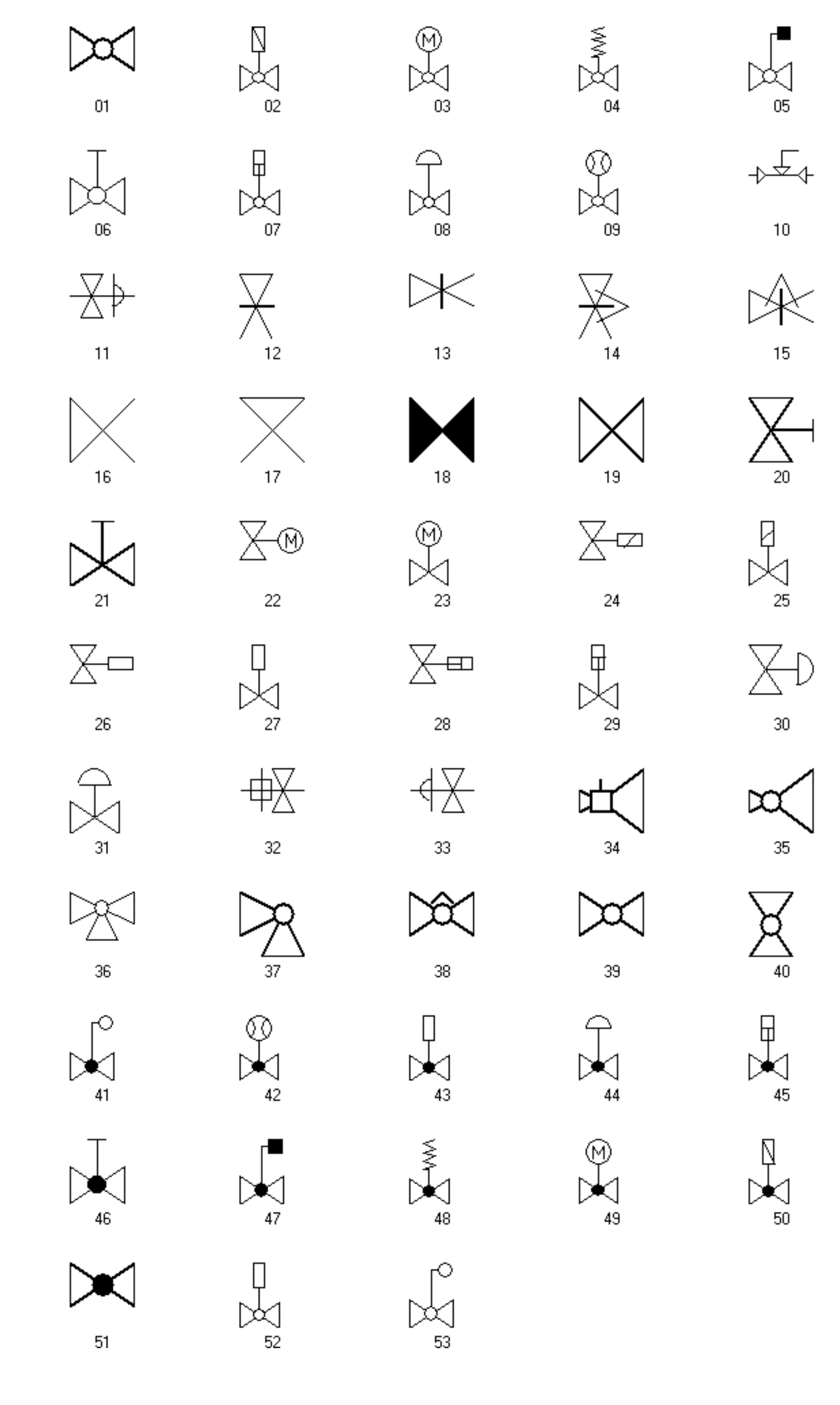


5.4.4.7 电机

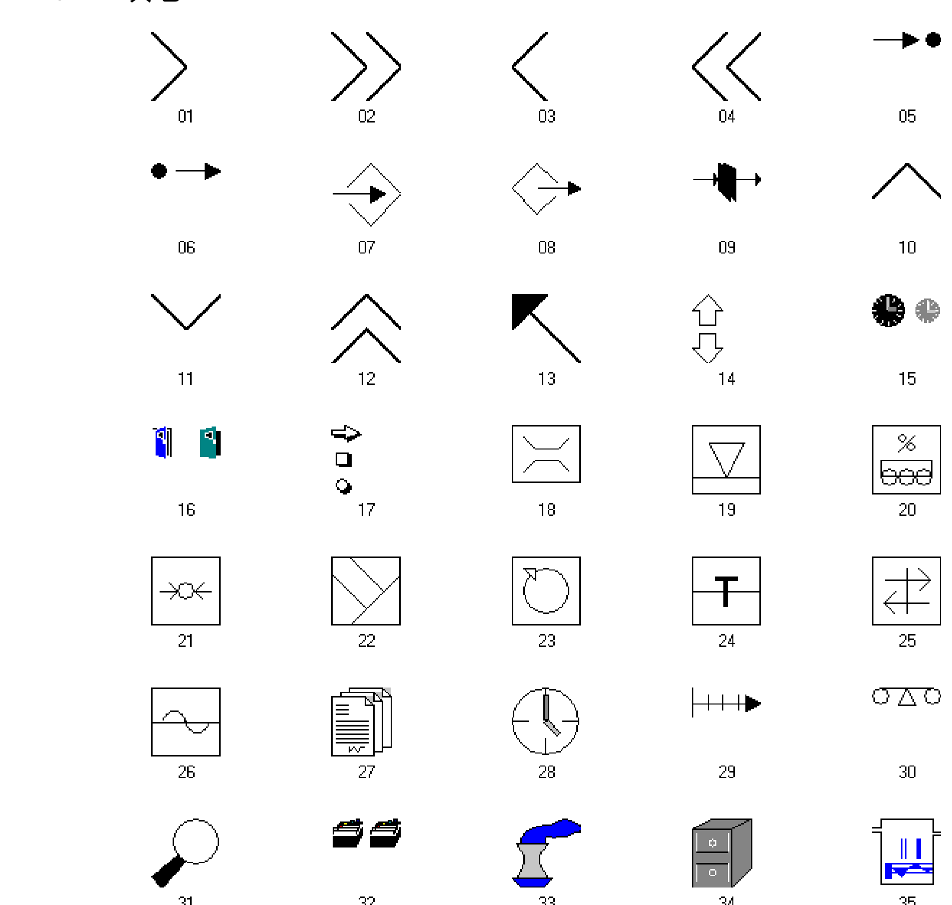




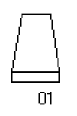
5.4.4.8 阀

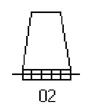

01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53

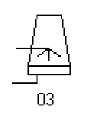
5.4.4.9 其它 1

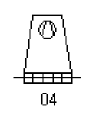

01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
%
21
22
23
24
T
25
26
27
28
29
30
31
32
33
34
35

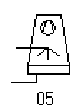
5.4.4.10 其它 2

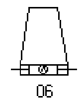

01

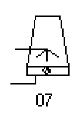

02

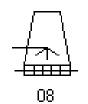

03

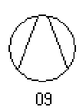

04

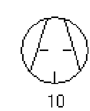

05

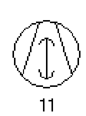

06

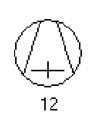

07

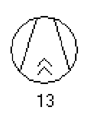

08

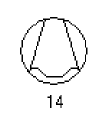

09

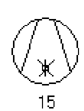

10

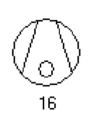

11

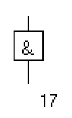

12


13

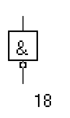

14


15

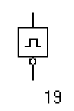

16

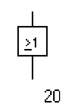


17

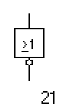


18

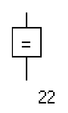

19



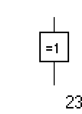
20



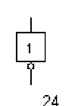
21



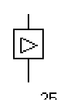
22

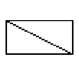


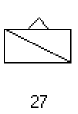
23

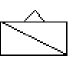


24


25


26


27


28

索引

D

F

Y

Z

S